Geometric and Information-Geometric Priors for Data-Driven Decision Making

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Niklas Jordi Freymuth

geb. in Oberhausen

Tag der mündlichen Prüfung: 13. Mai 2026
1. Referent: Prof. Dr. Gerhard Neumann
2. Referent: Prof. Dr. Carlo D'Eramo

Abstract

The rapid advancement of Machine Learning has transitioned from solving isolated, static tasks toward developing autonomous agents capable of operating in complex, dynamic environments. Imitation Learning (IL) and Reinforcement Learning (RL) provide the standard frameworks for such data-driven decision making, enabling agents to learn optimal behaviors through direct interaction or imitation of an expert. For these agents to succeed in real-world applications, they must actively interact with their surroundings and observe the resulting changes to achieve specific goals.

Yet, the efficacy of current frameworks is often bottlenecked by "black box" representations that do not exploit the spatial, topological, and distributional structures inherent in physical and optimization tasks. Standard methods tend to treat environment features and agent decisions as flat vectors or images, ignoring crucial underlying symmetries and constraints. We argue that Geometric Deep Learning (GDL) and Information Geometry (IG) provide the necessary structure for efficient, robust, and generalizable decision-making when integrated into autonomous agents' deployment and optimization.

This thesis addresses three primary challenges at the intersection of geometry and agentic behavior. We first consider the challenge of capturing versatile human behavior in IL, which is often hindered by absolute coordinate systems that make demonstrated and recorded human behavior appear inconsistent. Using geometric behavioral descriptors and information-geometric projections, our proposed method captures distinct modes of expert behavior, which enables agents to robustly generalize to novel contexts in versatile ways. Next, we consider the discretization schemes of traditional mesh-based physical simulation. By treating meshes as dynamic geometric graphs, we develop both RL and IL algorithms that reason over highly irregular domains. These methods automate adaptive meshing, focusing computational resources on critical regions for high-fidelity simulation. Finally, we move our attention to the optimization of agents in high-dimensional action spaces. We derive a trust-region method utilizing information-geometric projections that limit the update step size during training. This approach leverages the intrinsic manifold of the agent's behavior distribution to ensure that updates remain reliable over the course of training.

Together, these contributions demonstrate that grounding representation and optimization in geometric principles leads to more effective and stable learning. By aligning agent representations and optimization schemes with the geometry of the underlying problem, this work facilitates the development of more adaptable and structurally aware autonomous systems.

Zusammenfassung

Der rasante Fortschritt des maschinellen Lernens hat den Fokus von isolierten, statischen Aufgaben hin zu autonomen Agenten verschoben, die in komplexen und dynamischen Umgebungen agieren. Imitation Learning (IL) und Reinforcement Learning (RL) bieten hier einen effektiven Rahmen, um optimales Verhalten durch Interaktion oder Nachahmung zu erlernen. Für den realen Einsatz müssen diese Agenten aktiv mit ihrer Umgebung interagieren und resultierende Veränderungen gezielt nutzen, um ihre Ziele zu erreichen.

Existierende Ansätze scheitern oft an „Black-Box“-Repräsentationen, die die räumlichen, topologischen und strukturellen Eigenschaften physischer Aufgaben und der Optimierung nicht ausnutzen. Gängige Methoden behandeln Umgebungsmerkmale meist als einfache flache Vektoren oder Bilder und ignorieren dabei grundlegende Symmetrien und physikalische Randbedingungen. Wir argumentieren, dass Prinzipien aus dem Geometric Deep Learning (GDL) und der Informationsgeometrie (IG) die notwendige Struktur liefern, um eine effiziente, robuste und verallgemeinerbare Entscheidungsfindung zu ermöglichen.

Diese Arbeit adressiert drei zentrale Herausforderungen an der Schnittstelle zwischen Geometrie und dem Verhalten autonomer Agenten. Zuerst betrachten wir das Erlernen vielseitigen menschlichen Verhaltens im IL. Ein solches Lernen wird oft durch absolute Koordinatensysteme erschwert, die aufgenommene menschliche Demonstrationen unnötig inkonsistent erscheinen lassen. Durch den Einsatz geometrischer Verhaltensdeskriptoren und informationsgeometrischer Projektionen kann die von uns vorgestellte Methode verschiedene Expertenmodi präzise erfassen und so robust auf neue Kontexte generalisieren. Als Nächstes untersuchen wir Diskretisierungsschemata für die mesh-basierte physikalische Simulation. Indem wir Meshes als dynamische geometrische Graphen betrachten, entwickeln wir RL- und IL-Algorithmen, die auch auf hochgradig irregulären Domänen funktionieren. Diese Methoden automatisieren adaptives Meshing und konzentrieren damit Rechenressourcen dort, wo sie den größten Einfluss auf eine präzise Simulation haben. Schließlich widmen wir uns der Optimierung von Agenten in hochdimensionalen Entscheidungsräumen. Wir leiten ein Trust-Region-Verfahren ab, das den Optimierungsschritt mithilfe informationsgeometrischer Projektionen begrenzt. Dieser Ansatz nutzt den intrinsischen Raum der Verhaltensverteilung des Agenten, um stabile und zuverlässige Updateschritte während des gesamten Trainings zu garantieren.

Zusammenfassend zeigen diese Beiträge, dass die Einbettung geometrischer Prinzipien in Repräsentation und Optimierung die Effektivität und Stabilität von Lernprozessen maßgeblich steigert. Indem diese Arbeit Agentenmodelle und Optimierungsschemata konsequent an der Geometrie des zugrunde liegenden Problems ausrichtet, ebnet sie den Weg für die Entwicklung anpassungsfähigerer und strukturorientierterer autonomer Systeme.

Acknowledgements

I want to thank Geri for being an incredible academic advisor in every sense. Thank you for your continued guidance and for allowing me the freedom to explore my own ideas, while always providing sharp insights and critical feedback whenever I felt stuck. I deeply appreciate your patience and kindness, as well as your talent for cultivating a research group that is as supportive and kind as it is talented and productive. Most of the following acknowledgements are directed toward members of this group, which simply would not exist in this capacity without your leadership. Without you, this thesis would not have been possible.

Special thanks go out to my close friends and collaborators at the Autonomous Learning Robots lab. First and foremost, I thank Philipp Becker, who turned a successful master's thesis supervision into a long-lasting friendship. Thank you for always being there during the highs and lows of my PhD, for countless inspiring discussions, for being an academic role model throughout the whole journey of coming up with an idea, validating it, and writing the whole thing down. I appreciate your precise and cynical, yet ultimately uplifting view of the world, both within academia and beyond. Closely following are Nicolas Schreiber and Philipp "PhD" Dahlinger, who have been my daily companions throughout the last few years. Thank you, Nic, for sharing your encyclopedic knowledge of all things hardware, whether it was 3D printing, setting up a robot, or fixing a bike. Thank you, Philipp, for keeping things interesting with your endless well of niche hobbies and interests. I also want to thank Andreas Boltres for our friendship and lunch-date discussions over the years, and Balázs Gyenes for his dry, polite sense of humor and for always being available for a chat. You all taught me so much about research and life in general. Thank you for the boardgame nights, saving countless islands from those pesky invaders, and for all around making my time at the lab so much more enjoyable.

I owe my deepest gratitude to Tobias Würth for being the single best mechanical engineer a lowly computer science student could wish for. Without your expertise and saintly patience, many of the projects in this thesis would not have been possible. I also want to thank Johannes Mitsch and Luise Kärger for their knowledge on all things simulation, for providing important datasets and making sure that my work remained grounded in engineering principles. Next, I want to thank Aleksandar Taranovic and Tai Hoang for long-standing collaborations and fruitful discussions. Thank you, Aleks, for always being incredibly kind and sincere, and for being the glue that holds the lab together, and thank you, Tai, for your sharp mind and attention to detail. On a more fundamental level, the lab simply would not run without Christine Brand, who tirelessly and singlehandedly makes sure that we PhD students actually get to do research by protecting us from bureaucratic overload. Finally, I want to thank all other members of the ALR, both current and alumni, and all my collaborators over the years for making this experience what it is. Among these wonderful people are Onur Celik, Fabian Otto, Denis Blessing, Serge Thilges and Paul Scheikl. Thanks to all of you, I am one of the lucky few who can say that they truly enjoyed their PhD.

The structure and content of this work was greatly improved by the feedback of many colleagues and friends, both within and outside of the ALR. While some of you are already mentioned above, I want to thank Philipp Becker, Onur Celik, Philipp Dahlinger, Johannes Mitsch and Keke Merz for their valuable detailed feedback on different parts of the thesis.

I would also like to thank my family and friends outside of academia for their continued support throughout my studies. Foremost, I want to thank my parents for always believing in me and encouraging me to pursue my dreams. This PhD is a direct result of your confidence in my abilities and how you nurtured my curiosity and drive to ask questions. For that, I am forever grateful. Thank you also to my sister for being there for me, and for being one of the funniest and most genuine people I know. Finally, thank you to Valerie Horter for being my best friend, flatmate, and partner in crime throughout all these years. You fill many roles, and you do all of them amazingly well. There is no short way to express how much you mean to me.

Lastly, I want to thank Prof. Carlo D'Eramo and all other members of my PhD committee for their time and effort in reviewing this thesis.

Contents

List of Figures

List of Tables

1. Introduction

Driven by the widespread adoption of Large Language Model (LLM)-powered assistants and unprecedented industrial scaling, Machine Learning, and particularly Deep Learning (Goodfellow et al., 2016), has become one of the defining technologies of our time. Looking back, early applications of machine learning focused on isolated, static domains, and simple pattern recognition tasks. Classic examples include image classification (Deng et al., 2009; Krizhevsky et al., 2012), speech recognition (Hinton et al., 2012), and machine translation (Sutskever et al., 2014). Recent years have seen a shift toward autonomous systems operating in increasingly complex, dynamic environments. The examples above have evolved into more interactive and adaptive applications, often combining multiple types of data and requiring real-time decision-making (Radford et al., 2021; Shao et al., 2024). These advances similarly give rise to capable embodied agents that can translate verbal instructions into precise robotic motor commands (Kim et al., 2024; Intelligence et al., 2025). To succeed, such systems must actively interact with their surroundings and observe the resulting changes to achieve specific goals. This paradigm shift aligns with the hypothesis that intelligence is deeply rooted in physical grounding and interaction (Brooks, 1991), a perspective that has recently resurfaced in research on embodied generalist agents (Silver et al., 2021). The question, then, becomes how to best represent and model such complex, dynamic environments, and the system's interaction with them.

Reinforcement Learning (RL) (Sutton et al., 2018) and its demonstration-driven counterpart, Imitation Learning (IL) (Argall et al., 2009), have emerged as promising frameworks for training agents to operate in such environments. By framing a given problem as a sequential decision-making task, these approaches enable agents to learn optimal behaviors through interactions with the environment, either via trial and error or by imitating expert demonstrations. In both cases, the goal is to learn a policy, which is a mapping from an environment state to an action, that maximizes some utility function, often expressed as a reward signal in RL or implicitly defined by expert behavior in IL. RL excels in scenarios where a clear objective exists but the optimal way to achieve this objective is unknown, rendering it particularly suitable for playing games with clearly defined rules (Mnih et al., 2015; Silver et al., 2017; Vinyals et al., 2019). However, RL struggles in settings where such a reward signal is difficult to define, and may require extensive exploration of the environment to find effective strategies. Conversely, IL bypasses the need for reward engineering by leveraging existing expert knowledge, but is constrained by the quality and coverage of the provided demonstrations (Osa et al., 2018). The choice between RL and IL thus comes down to the availability of accurate and expressive reward models versus the accessibility and quality of demonstration data.

Both RL and IL methods aim to learn an effective policy. However, their success depends heavily on the structure of the underlying environment and action representations. In more complex domains, such as robotic manipulation or adaptive physical simulation, agents are often faced with high-dimensional state-action spaces that are difficult to understand and traverse. For example, IL from human data must contend with the inherent versatility of human behavior, while working with adaptive physical systems requires reasoning about the interplay between local features and overall simulation behavior on irregular domains. To make matters worse, standard methods often treat environment features and agent decisions as flat vectors or rasterized, fixed-resolution images (Mnih et al., 2015; Sutton et al., 2018). By ignoring the inherent geometric structure and topological constraints that govern real-world dynamics, these methods require vast datasets, handcrafted heuristics, or extensive manual design, which limits adaptability and scalability. This lack of structural awareness also extends to the optimization itself, as standard algorithms only superficially account for the intrinsic geometry of the policy space during updates. A key challenge in developing more effective agents thus lies in moving beyond ”black-box” mappings by utilizing representations and objectives that naturally encode the geometric structure of the task.

Geometric Deep Learning (GDL) (Bronstein et al., 2021) offers a principled way to incorporate geometric priors into learning architectures, significantly enhancing data efficiency, generalization, and robustness of the learned models. More broadly, Information Geometry (IG) (Amari, 2016) provides a theoretical framework for understanding the geometry of probability distributions, which can be leveraged to design more stable and efficient optimization algorithms. This thesis investigates integrating both principles within RL and IL frameworks. Grounding the learning process in the geometry of the underlying task allows us to organize and simplify otherwise unstructured search spaces. Similarly, introducing information-geometric inductive biases into the optimization process can stabilize the learning process and guide agents toward more robust solutions. To this end, we focus on three distinct challenges related to geometric and information-geometric priors for learning agents and their data-driven decision-making process.

First, we consider the interaction with our real-world environment through the lens of geometric representations. Capturing versatile human behavior presents a core challenge for IL in robotic manipulation. Humans intuitively solve tasks on a semantic level, internally representing goals and constraints in relation to their surroundings, rather than considering absolute positions and orientations. For example, when reaching for a cup, we focus on the relative position of the cup to our hand and the overall geometry of the task, rather than the specific joint angles or absolute coordinates of our arm. Further, they automatically factor in symmetries, such as reaching around an object from either side, or preferring one hand over the other. Consequently, viewing human demonstrations in terms of absolute coordinate systems makes them appear inherently multi-modal, as, given the same goal, the actual taken trajectories vary drastically based on subtle contextual cues or personal preference. This mismatch hinders learning human-like behaviors for many learning algorithms, which often expect consistent, unimodal data.

Second, we explore the role of geometry in the simulation of physical systems, specifically concerning the discretization of physical domains. Accurate simulation of complex physics is crucial to engineering applications. Similarly, an accurate understanding of the physical world and the ability to simulate consequences of interactions with this world are prerequisites for more advanced embodied robotic and reasoning tasks. Numerical methods such as the Finite Element Method (FEM) (Brenner et al., 2008) enable such simulations by discretizing the physical domain into a mesh comprised of many simple elements. This discretization comes with a trade-off between computational efficiency and accuracy, as both the accuracy and computational cost of a simulation depend on the local density of the underlying mesh. Adaptive meshing (Plewa et al., 2005) dynamically adapts local mesh resolution to provide more computational resources to regions that are critical for a given simulation. By providing a locally refined mesh, adaptive meshing therefore allows for significantly improved simulation accuracy while keeping computational costs manageable. However, traditional methods often use hand-tuned, problem-specific heuristics or rely on costly error calculations that neither generalize nor scale well, often making their practical application difficult (Mukherjee, 1996; Kita et al., 2001; Wallwork, 2021). Learning-based adaptive meshing is a promising alternative but must handle the mesh as a dynamic geometric entity rather than a fixed grid. When aiming to optimize meshes based on direct simulation outcomes via RL, learned methods must effectively integrate localized feedback, like element-wise quality estimates, rather than relying on global, predefined reward functions. Conversely, learning to imitate expert-designed meshes with IL requires reasoning about the domain topology and finding relevant patterns for local mesh adaptation decisions. Both cases require discerning the complex, often non-linear, interplay between local geometric features and overall simulation behavior on irregular, spatially varying domains.

Third, we investigate the influence of geometric priors on the optimization processes underlying the training of learning agents. While the previous challenges address the geometry of the external environment, this challenge concerns the internal geometry of the agent's decision-making process, i.e., the manifold of its policy distribution. In many RL settings, optimizing this policy distribution, i.e., the distribution over possible actions in a given state, relies on feedback gathered from trajectories sampled by the current policy. While this feedback makes for a relatively accurate local approximation of the target objective, it quickly becomes unreliable when the gathered data no longer represents the updated policy's behavior. To prevent catastrophic collapses in performance, especially in high-dimensional action spaces, we must ensure that policy updates stay in a reliable local region. We therefore ask how we can leverage IG to enforce a notion of proximity between the current policy and the one used to gather the data.

In this thesis, we address these challenges by developing novel frameworks that integrate geometric priors and information-geometric principles into the perception, representation, and optimization of learning agents. We propose geometrically encoding tasks and environments to enable agents to learn more effectively from multimodal demonstrations. Next, we present novel RL and IL algorithms that inherently respect the graph-based geometric structure of physical domains. These methods facilitate efficient domain discretization via adaptive meshing, ensuring that computational resources are focused where they are most needed for high-fidelity physical simulation. Finally, we derive a trust-region optimization method that utilizes information-geometric constraints to stabilize the RL training process in high-dimensional action spaces.

1.1. Thesis Contributions and Structure

We summarize the above challenges into three main research questions that we address in this thesis:

Q1 How can we utilize geometric representations to capture versatile human expert behavior in robotic manipulation? (Chapter 3)

Q2 How can we enable learning agents to leverage intrinsic geometric features to represent, reason over, and optimize the discretization of irregular physical domains? (Chapter 4 and Chapter 5)

Q3 How can we employ information-geometric constraints to stabilize policy optimization in high-dimensional action spaces? (Chapter 6)

Structure of the Thesis. Chapter 2 introduces the necessary background for the thesis, covering Reinforcement Learning (RL) and Imitation Learning (IL) as agent-centric learning paradigms. It also introduces the principles of Geometric Deep Learning (GDL) underlying our geometric representations and architectures, and briefly covers the basics of Information Geometry (IG) relevant to our optimization methods. Table 1.1 provides an overview of the methods developed in this thesis and their underlying paradigms.

Chapter	**3**	**4**	**5**	**6**
Method	*VIGOR*	*ASMR*	*AMBER*	*TROLL*
Learning Paradigm	IL	RL	IL	RL
Information-Geometric Optimization	×			×
Geometric Priors	×	×	×	

Table 1.1.: Overview of the specific methods developed in this thesis and the paradigms they utilize.

Chapter 3 addresses the first research question by introducing Versatile Imitation from Geometrically Observed Representations (*VIGOR*), a novel IL method based on geometric behavioral descriptors that encode task states as geometric relations. Within the resulting feature space, trajectories of similar performance appear similar regardless of specific task configurations, allowing for sample-efficient generalization to novel contexts. *VIGOR* employs the Information-Projection, which is the geometric minimization of the relative entropy from the learner's policy to the expert's distribution. The presented method additionally constrains individual updates based on the proximity to the previous policy to stabilize the optimization. This combination allows training expressive trajectory-level Gaussian Mixture policies that capture distinct expert behaviors.

Next, we explore the second research question in Chapter 4 and Chapter 5, where we investigate how learning agents can optimize the discretization of physical domains. These chapters address two distinct axes of the problem. On one axis, we examine the learning paradigm, comparing active interaction via RL with the imitation of expert strategies through IL. On the other axis, we consider the nature of the adaptation itself, contrasting Adaptive Mesh Refinement (AMR), which iteratively refines existing elements, with Adaptive Mesh Generation (AMG), which utilizes a continuous resolution field over the domain to generate a novel mesh. Independent of the specific learning paradigm and adaptation method, these approaches must inherently respect and manipulate the underlying geometry of the physical domain. This domain often consists of irregular topologies and, by the nature of adaptive meshing, must be considered on varying local resolutions that render classical grid-based approaches ineffective. We therefore view the mesh as a geometric graph, allowing us to leverage principles from GDL to design architectures and optimization schemes that remain effective across diverse and evolving geometric structures. Ultimately, this approach exploits the problem geometry on two levels. A Graph Neural Network (GNN) architecture naturally processes the irregular structure of the considered domain, while the iterative meshing procedure allows the policy to decide the resolution of the intermediate meshes on which it subsequently predicts.

Chapter 4 combines RL with AMR to develop Adaptive Swarm Mesh Refinement (*ASMR*), which formulates the refinement process as a multi-agent coordination problem. We represent each mesh element as one of many autonomous and homogeneous agents that must decide whether to mark themselves for refinement, i.e., subdivision, to improve the overall simulation accuracy, or to stay unrefined and save computational resources. To handle the previously mentioned evolution in geometric structure, which presents itself here as a varying number of agents, we develop a framework that remains stable even as the agent population changes over time. Within this framework, we equip the individual agents with localized rewards that facilitate refinements in regions of high simulation error. To ensure consistent optimization across multiple refinement steps from these local rewards, we construct a mapping of agents over time that allows agents in early steps to learn from the feedback gathered by their successors.

Conversely, Adaptive Meshing By Expert Reconstruction (*AMBER*) in Chapter 5 addresses Adaptive Mesh Generation (AMG) via IL by developing a method that learns to predict a local mesh resolution directly from expert examples. Starting from an arbitrary mesh, we train a policy to predict a target resolution for each element and then generate a new mesh that adheres to this target resolution. Since the initial mesh discretizes the continuous resolution space, we perform this prediction iteratively, gradually refining the mesh toward the design of the expert. By training on a replay buffer of these self-generated intermediate states, the agent learns from its own evolving data distribution, ensuring that its policy remains robust across a sequence of mesh generation steps. To effectively reason about the resulting adaptive geometric structure, we employ a hierarchical GNN architecture that captures both local features and global context. This hierarchy enables consistent communication across the entire domain, allowing the policy to make informed local decisions regardless of the current mesh density.

Finally, Chapter 6 addresses the third research question with Trust Region Optimization for Large Language models (*TROLL*), investigating the influence of information-geometric constraints on the optimization of learning agents. While the previous chapters focus on the geometry of the external environment, this chapter is about the internal geometry of the decision-making process, specifically the manifold of the policy distribution. In high-dimensional action spaces, such as those encountered when fine-tuning LLMs, policy optimization must be carefully constrained to ensure stable learning. We address this issue by utilizing an information-geometric trust-region projection that keeps the optimized policy close to the one used to gather the data. We then briefly combine the resulting RL objective with the framework of Chapter 4, demonstrating its efficacy in the context of adaptive meshing. This combination showcases how leveraging the intrinsic geometry of the policy space enables stable and efficient optimization across diverse problem domains.

Finally, Chapter 7 concludes the thesis with a summary of our contributions and a discussion on the future of geometry-aware learning agents. *The following four sections are reprints of the abstracts of the respective publications this thesis is based on.*

1.1.1. Inferring Versatile Behavior from Demonstrations by Matching Geometric Descriptors

Niklas Freymuth, Nicolas Schreiber, Philipp Becker, Aleksandar Taranovic, Gerhard Neumann (2022), Chapter 3

Humans intuitively solve tasks in versatile ways, varying their behavior in terms of trajectory-based planning and for individual steps. Thus, they can easily generalize and adapt to new and changing environments. Current Imitation Learning algorithms often only consider unimodal expert demonstrations and act in a state-action-based setting, making it difficult for them to imitate human behavior in case of versatile demonstrations. Instead, we combine a mixture of movement primitives with a distribution matching objective to learn versatile behaviors that match the expert's behavior and versatility. To facilitate generalization to novel task configurations, we do not directly match the agent's and expert's trajectory distributions but rather work with concise geometric descriptors which generalize well to unseen task configurations. We empirically validate our method on various robot tasks using versatile human demonstrations and compare to imitation learning algorithms in a state-action setting as well as a trajectory-based setting. We find that the geometric descriptors greatly help in generalizing to new task configurations and that combining them with our distribution-matching objective is crucial for representing and reproducing versatile behavior.

1.1.2. Adaptive Swarm Mesh Refinement using Deep Reinforcement Learning with Local Rewards

Niklas Freymuth, Philipp Dahlinger, Tobias Würth, Simon Reisch, Luise Kärger, Gerhard Neumann (2024), extending Niklas Freymuth et al. (2023), Chapter 4

Simulating physical systems is essential in engineering, but analytical solutions are limited to straightforward problems. Consequently, numerical methods like the Finite Element Method (FEM) are widely used. However, the FEM becomes computationally expensive as problem complexity and accuracy demands increase. Adaptive Mesh Refinement (AMR) improves the FEM by dynamically placing mesh elements on the domain, balancing computational speed and accuracy. Classical AMR depends on heuristics or expensive error estimators, which may lead to suboptimal performance for complex simulations. While AMR methods based on machine learning are promising, they currently only scale to simple problems. In this work, we formulate AMR as a system of collaborating, homogeneous agents that iteratively split into multiple new agents. This agent-wise perspective enables a spatial reward formulation focused on reducing the maximum mesh element error. Our approach, Adaptive Swarm Mesh Refinement++ (ASMR++), offers efficient, stable optimization and generates highly adaptive meshes at user-defined resolution at inference time. Extensive experiments demonstrate that ASMR outperforms heuristic approaches and learned baselines, matching the performance of expensive error-based oracle AMR strategies. ASMR additionally generalizes to different domains during inference, and produces meshes that simulate up to 2 orders of magnitude faster than uniform refinements in more demanding settings.

1.1.3. AMBER: Adaptive Mesh Generation by Iterative Mesh Resolution Prediction

Niklas Freymuth, Tobias Würth, Nicolas Schreiber, Balazs Gyenes, Andreas Boltres, Johannes Mitsch, Aleksandar Taranovic, Tai Hoang, Philipp Dahlinger, Philipp Becker, Luise Kärger, Gerhard Neumann (2025), Chapter 5

The cost and accuracy of simulating complex physical systems using the Finite Element Method (FEM) scales with the resolution of the underlying mesh. Adaptive meshes improve computational efficiency by refining resolution in critical regions, but typically require task-specific heuristics or cumbersome manual design by a human expert. We propose Adaptive Meshing By Expert Reconstruction (AMBER), a supervised learning approach to mesh adaptation. Starting from a coarse mesh, AMBER iteratively predicts the sizing field, i.e., a function mapping from the geometry to the local element size of the target mesh, and uses this prediction to produce a new intermediate mesh using an out-of-the-box mesh generator. This process is enabled through a hierarchical graph neural network, and relies on data augmentation by automatically projecting expert labels onto AMBER-generated data during training. We evaluate AMBER on 2D and 3D datasets, including classical physics problems, mechanical components, and real-world industrial designs with human expert meshes. AMBER generalizes to unseen geometries and consistently outperforms multiple recent baselines, including ones using Graph and Convolutional Neural Networks, and Reinforcement Learning-based approaches.

1.1.4. TROLL: Trust Regions improve Reinforcement Learning for Large Language Models

Philipp Becker, Niklas Freymuth*, Serge Thilges, Fabian Otto, Gerhard Neumann (2026), Chapter 6*

Reinforcement Learning (RL) with PPO-like clip objectives has become the standard choice for reward-based fine-tuning of large language models (LLMs). Although recent work has explored improved estimators of advantages and normalization, the clipping mechanism itself has remained untouched. Originally introduced as a proxy for principled KL-based trust regions, clipping is a crude approximation that often causes unstable updates and suboptimal performance. We replace the clip objective with a novel discrete differentiable trust region projection, which provides principled token-level KL constraints. The projection operates on a sparse subset of the model's most important token logits to balance computational cost and projection effectiveness. Our approach, Trust Region Optimization for Large Language Models (TROLL), serves as a direct replacement for PPO-like clipping during training and does not alter the model's inference behavior. Across mathematical reasoning and code generation tasks, model families, as well as advantage-estimation methods, TROLL consistently outperforms PPO-like clipping in terms of training speed, stability, and final success rates.

* Equal contribution

2. Foundations

This chapter covers the theoretical foundations for this thesis. Each section introduces a set of concepts, methods, and tools that we build upon, and refers to the chapter(s) to which it is relevant. Concretely, Section 2.1 covers Imitation Learning (IL) and Reinforcement Learning (RL), which provide the core framework for developing decision-making agents from demonstrations and reward signals. It also introduces Variational Inference (VI) to frame these decision-making problems as the optimization of distributional objectives over the space of policies. As the stability of these methods often relies on the geometry of the policy space, Section 2.2 introduces key concepts from information geometry that let us reason over probability distributions as points on a manifold. From this viewpoint, we derive trust-region methods for stable policy optimization. Shifting to physical geometry, Section 2.3 introduces the Finite Element Method (FEM) as a tool for simulating complex physics and as an application domain for geometric inductive biases for data-driven learning. Considering a learning setting, the section then covers Geometric Deep Learning (GDL), particularly symmetry principles and Graph Neural Network (GNN) architectures. In doing so, it provides the tools to model and learn over irregular, graph-structured data, including unstructured meshes for the FEM.

Notation. Throughout the thesis, we denote scalars as regular letters, e.g., x, vectors using bold lowercase letters, e.g., $\mathbf{x}$, and matrices using bold uppercase letters, e.g., $\mathbf{X}$. We denote the proposed approaches and their baselines in *italics*, and tasks, environments, and general experiment names are set in `teletype font`.

2.1. Learning for Decision-Making

We start by introducing the theoretical foundations of learning for data-driven decision-making, which is the core framework for the approaches proposed in this thesis. In particular, we are interested in the ability to learn from sequences of interactions and to optimize behavior based on demonstrations and non-differentiable environment feedback, as well as under stability constraints. We first consider Imitation Learning (IL), which enables learning from expert demonstrations, and then Reinforcement Learning (RL), which provides a framework for optimizing behavior through environmental interaction and an explicit reward signal. Both paradigms can be viewed through the lens of distribution matching, and are closely connected to Variational Inference (VI), which provides a principled framework for approximating complex target distributions. This distributional perspective is central to the policy optimization methods developed in this thesis, as it allows us to leverage information-theoretic principles for stable and efficient learning. The following sections briefly summarize fundamental concepts and algorithms in IL, RL, and VI that are relevant to this thesis. Unless mentioned otherwise, the summary of IL and RL is based on Argall et al. (2009) and Sutton et al. (2018), respectively, while the overview of VI is based on Blei et al. (2017).

2.1.1. Imitation Learning

A natural starting point for training decision-making agents from data is to leverage existing expertise. In many real-world scenarios, a human operator or an automated system can already perform the desired task, and the challenge is to transfer this capability to a learning agent. IL addresses this challenge by enabling agents to learn directly from a dataset $\mathcal{D}_E = \{\tau_i\}_{i=1\ldots N}$ of expert demonstrations without the need for an explicit reward signal. Learning from demonstrations is particularly useful when the task involves complex skills or preferences that are hard to quantify as a scalar objective. In the context of this thesis, IL is employed for learning from versatile human behavior in Chapter 3, and imitating expert meshes in Chapter 5.

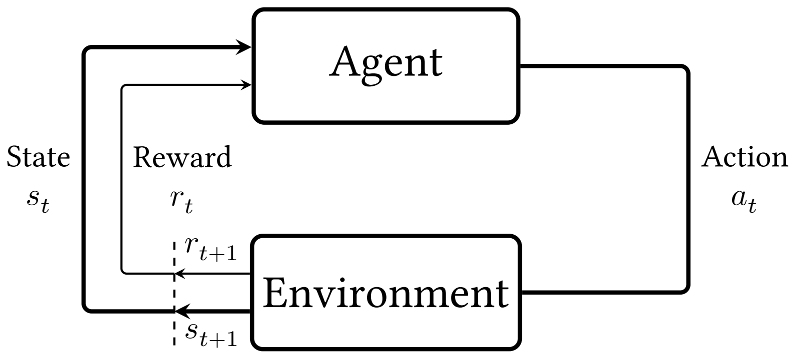


Figure 2.1.: Agent-Environment interaction loop in a Markov Decision Process. When presented with a state s_t, the agent selects an action a_t that is executed in the environment, which then returns a new state s_{t+1} and potentially a reward signal r_{t+1} to the agent. Adapted from Sutton et al. (2018).

Modeling Sequential Decision-Making. The theoretical foundation for sequential decision-making is the Markov Decision Process (MDP) (Bellman, 1957), which models the interaction between an agent and its environment as a discrete-time stochastic process. Formally, an MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, p, p_0, r, \gamma)$. Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space and $p(s'|s, a)$ is the transition probability of moving from state s to state s' after taking action a. The set $p_0 \subseteq \mathcal{S}$ defines the distribution over initial states. The reward function $r(s, a) \in \mathbb{R}$ assigns a scalar reward to action a taken in state s, and the discount factor $\gamma \in [0, 1]$ determines the importance of future rewards. While the reward function is not directly used in all IL methods, it plays a central role in RL and in discriminator-based IL approaches discussed below. Figure 2.1 illustrates the interaction loop between the agent and the environment in an MDP.

Policies and Trajectories. An agent interacts with the MDP by following a policy $\pi(a|s)$, which defines a distribution over actions conditioned on the current state. The resulting sequence of states and actions forms a trajectory $\tau = (s_0, a_0, s_1, a_1, \dots)$, where $s_0 \sim p_0$ and each subsequent state is sampled according to the transition dynamics $p(s'|s,a)$. In our case, the policy π is usually parameterized by a vector θ describing the weights of a neural network, denoted π_θ. To reason about the long-term behavior of a policy, we define the discounted state-action occupancy measure

$$\rho^\pi(s,a) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi), \tag{2.1}$$

where $P(s_t = s|\pi)$ denotes the probability of being in state s at time t when following π. This measure captures the frequency with which a policy visits specific state-action pairs, and is central to both the IL and RL objectives discussed in the following.

Behavioral Cloning. The most straightforward approach to IL is Behavioral Cloning (BC), which treats imitation as a supervised learning problem. Given either expert trajectories or state-action pairs, the agent's policy is trained to minimize a supervised loss function, such as the cross-entropy or Mean Squared Error (MSE), to match the expert's actions. While computationally efficient and simple to implement, BC suffers from the problem of covariate shift. Because the agent is only trained on states visited by the expert, small execution errors can lead the agent into unfamiliar states where it has no training data, causing errors to compound over the course of a trajectory.

Interactive Distribution Matching. To mitigate the compounding errors of offline methods like BC, a family of interactive algorithms (Ross et al., 2011) proposes incorporating the expert in the training loop. By executing the current agent policy to collect new trajectories and subsequently querying the expert for the optimal actions in those states, the training distribution is iteratively shifted to match the states actually visited by the agent. This online data generation ensures that the agent learns how to recover from its own mistakes, which is a powerful mechanism for improving policy robustness when the expert is easy to query. However, it requires repeated access to an often costly expert, which may not be feasible in many real-world scenarios.

Discriminator-Based Imitation Learning. An alternative perspective is to treat IL as a distribution matching problem, where the goal is to make the agent's occupancy measure $\rho^{\pi_\theta}(s,a)$ indistinguishable from that of the expert $\rho^{\pi_E}(s,a)$. Frameworks such as Generative Adversarial Imitation Learning (GAIL) (Ho et al., 2016) employ a discriminator-based objective where a binary classifier $\boldsymbol{\psi} : \mathcal{S} \times \mathcal{A} \to (0,1)$ is trained to distinguish between expert and agent transitions. Formally, GAIL solves a minimax game where the discriminator $\boldsymbol{\psi}$ and the policy π_θ are optimized according to

$$\min_{\pi_\theta} \max_{\boldsymbol{\psi}} \mathbb{E}_{(s,a)\sim\rho^{\pi_\theta}}[\log(1 - \boldsymbol{\psi}(s,a))] + \mathbb{E}_{(s,a)\sim\rho^{\pi_E}}[\log \boldsymbol{\psi}(s,a)] - \lambda H(\pi_\theta), \tag{2.2}$$

where $H(\pi_\theta)$ is a causal entropy regularization term and π_E denotes the expert policy, which is only available in the form of demonstrations. Since the expert distribution is only implicitly defined through the dataset, the discriminator signal serves as the primary learning signal for the agent, effectively replacing the reward function of the MDP with a learned surrogate. The agent then uses RL methods, as detailed in the following section, to optimize its policy against this surrogate reward. A common choice is $r(s,a) = -\log(1 - \psi(s,a))$, which encourages the agent to take actions that the discriminator classifies as expert-like. In its basic form, this setup is framed in an adversarial manner, similar to, e.g., Generative Adversarial Networks (Goodfellow et al., 2014), where the agent is essentially tasked to fool the discriminator. At convergence, the agent's policy is indistinguishable from the expert's behavior, as the discriminator can no longer differentiate between them.

2.1.2. Reinforcement Learning

While IL leverages existing demonstrations, it requires access to high-quality expert data, which may not always be available. An alternative is to define an explicit reward signal that quantifies the desired behavior, and to let the agent discover optimal strategies through interaction with the environment. RL provides this framework, and is central to the multi-agent mesh refinement strategies in Chapter 4 and the high-dimensional trust-region methods in Chapter 6.

Reinforcement Learning Objective. The goal of RL is to find a policy that maximizes the expected cumulative discounted reward, also known as return. The probability of a trajectory τ occurring under a policy π is determined by both the policy and the environment dynamics, i.e., $P(\tau|\pi) = p_0(s_0) \prod_{t=0}^{\infty} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t)$. The return of a trajectory is denoted as $R(\tau) = \sum_{k=0}^{\infty} \gamma^k r(s_k, a_k)$, and we are usually interested in maximizing its expectation, i.e.,

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] = \int P(\tau|\pi)R(\tau)d\tau. \tag{2.3}$$

Maximizing the return corresponds to finding optimal parameters $\theta^* = \arg\max_\theta J(\pi_\theta)$ for the policy π_θ. The key conceptual insight for MDPs is that the system is Markovian, meaning the future is independent of the past given the present state. This property gives rise to several powerful tools for analyzing and optimizing policies, such as value functions and the Bellman equations, which we will discuss in the following.

Value Functions and Bellman Optimality. A crucial tool for evaluating the quality of a given state or action is the concept of value functions. Most common are the state-value function $V^{\pi_\theta}(s)$ and the state-action-value function $Q^{\pi_\theta}(s,a)$, often called Q-function, both of which represent the expected return under a given policy π_θ from a specific starting point. To simplify notation, let $\tau_{t:} = (s_t, a_t, s_{t+1}, a_{t+1}, \dots)$ denote a trajectory tail starting from time t. The state-value function $V^{\pi_\theta}(s)$ is the expected return over all trajectories starting in state s, i.e.,

$$V^{\pi_\theta}(s) = \mathbb{E}_{\tau_{t:} \sim \pi_\theta} \left[R(\tau_{t:}) \mid s_t = s \right].$$

Similarly, the Q-function $Q^{\pi_\theta}(s, a)$ evaluates the expected return starting from state s and taking action a, with all subsequent steps following policy π_θ:

$$Q^{\pi_\theta}(s, a) = \mathbb{E}_{\tau_{t:} \sim \pi_\theta} \left[R(\tau_{t:}) \mid s_t = s, a_t = a \right].$$

Both functions can be expressed in terms of each other via the Bellman equations, i.e.,

$$V^{\pi_\theta}(s) = \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[Q^{\pi_\theta}(s, a)]$$

and

$$Q^{\pi_\theta}(s, a) = \mathbb{E}_{s' \sim p(\cdot|s,a)}[r(s, a) + \gamma V^{\pi_\theta}(s')].$$

By combining these two equations, we obtain the recursive form of the Bellman equation for the state-value function:

$$V^{\pi_\theta}(s) = \mathbb{E}_{a \sim \pi_\theta, s' \sim p(\cdot|s,a)}[r(s, a) + \gamma V^{\pi_\theta}(s') \mid s_t = s].$$

From here, the concept of Bellman optimality can be derived. Intuitively, Bellman optimality states that an overall optimal policy is composed of locally optimal decisions. If we behave optimally in the current state and continue to do so for all future states, the resulting trajectory is globally optimal. Formally, we define the optimal value functions $V^*(s) = \max_\pi V^\pi(s)$ and $Q^*(s, a) = \max_\pi Q^\pi(s, a)$, which satisfy the Bellman optimality equations

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a) \tag{2.4}$$

and

$$Q^*(s, a) = \mathbb{E}_{s' \sim p(\cdot|s,a)}[r(s, a) + \gamma V^*(s')]. \tag{2.5}$$

For small, tabular MDPs with known transition dynamics, these equations can be solved exactly using dynamic programming methods. However, for the more complex state and action spaces considered in this thesis, we must rely on function approximation methods to represent our policies and value functions. Among these, policy gradient methods are particularly popular and effective, as they allow for the direct optimization of the policy parameters.

Policy Gradient Methods. Unlike value-based methods that derive a policy from $Q^*(s, a)$, policy gradient methods (Sutton et al., 1999) directly optimize the policy parameters θ by following the gradient of the objective $J(\pi_\theta)$ with respect to θ. Policy gradient methods utilize the log-derivative trick, given as $\nabla_\theta P(\tau|\pi_\theta) = P(\tau|\pi_\theta) \nabla_\theta \log P(\tau|\pi_\theta)$, to derive the eponymous policy gradient

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_\theta \log P(\tau|\pi_\theta) R(\tau) \right] = \mathbb{E}_{\tau \sim \pi_\theta} \left[\left(\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) R(\tau) \right],$$

where the environment dynamics terms cancel out in the second equality as they do not depend on θ. The resulting gradient estimator is unbiased, and directly applying it via stochastic gradient ascent yields the REINFORCE algorithm (Williams, 1992), the foundational policy gradient method. While REINFORCE is a complete and functional learning algorithm, it suffers from high variance because the total reward $R(\tau)$ is used to evaluate every action in the trajectory. As a remedy, we can use the causal structure of our environment and replace the trajectory reward $R(\tau)$ with a single-sample estimate of the Q-Function, the reward-to-go $R(\tau_{t:}) = \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k)$, which only considers rewards obtained after time t. This substitution yields a lower-variance version of the policy gradient, given as

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau_{t:})\right].$$

Despite this improvement, the above objective is relatively unstable because $R(\tau_{t:})$ is a single-sample realization of the return that depends on all subsequent state transitions and future action selections within the episode. To further reduce the variance, we first realize that the expectation of the policy gradient remains unchanged when subtracting an arbitrary baseline $b(s_t)$ that is independent of the current action.

$$\mathbb{E}_{a_t\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)] = b(s_t)\nabla_\theta \mathbb{E}_{a_t\sim\pi_\theta}[1] = b(s_t)\nabla_\theta 1 = 0.$$

Here, we reversed the log-derivative trick to move the gradient outside the expectation, and used the fact that the probability distribution integrates to one. A common choice for this baseline is the state-value function $V^{\pi_\theta}(s_t)$, as it represents the average return the agent expects to receive from that state. Using the value function as a baseline lets us reformulate the policy gradient as (Mnih et al., 2016)

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right],$$

which uses the advantage function

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s), \tag{2.6}$$

which is derived from $R(\tau_{t:})$ being a single-sample estimate of the Q-Function. Intuitively, the advantage quantifies the relative benefit of a specific action at a given time step compared to the average behavior of the policy. While the above derivation uses trajectory-level expectations, we can equivalently express the policy gradient using the occupancy measure of Equation 2.1, yielding an expectation over state-action pairs rather than full trajectories:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{(s,a)\sim\rho^{\pi_\theta}}\left[\nabla_\theta \log \pi_\theta(a|s) A^{\pi_\theta}(s, a)\right].$$

This per-step formulation is a direct consequence of the policy gradient theorem (Sutton et al., 1999) and forms the basis of actor-critic methods (Mnih et al., 2016). In these methods, a separately learned value function, i.e., the critic, provides the baseline for estimating the advantage of each action selected by the policy, i.e., the actor. The policy gradient framework extends naturally beyond physical environments to any sequential decision-making problem that can be modeled as an MDP. A particularly impactful recent application is the post-training of Large Language Models (LLMs), where the sequential generation of tokens can be framed as a policy optimization problem.

Large Language Models as Reinforcement Learning Agents. RL for Large Language Model (LLM) post-training is used to align pre-trained text generation models with human preferences (Rafailov et al., 2023), and to improve their reasoning capabilities in logical domains (Shao et al., 2024). The latter is often called Reinforcement Learning from Verifiable Rewards (RLVR) and is concerned with solving complex math and code generation problems. We focus on this setting for the trust-region projections utilized in Chapter 6. While the techniques and algorithms used in RL for LLM post-training do not fundamentally differ from other domains, the size of the models, their pre-training and the structure of the considered problems introduce several unique challenges.

To apply the above framework in this setting, the generation process is cast as an MDP over tokens rather than physical states and actions. Following popular practice (Shao et al., 2024), we denote the query as $\mathbf{q}$ and the generated token sequence prior to step t as $\mathbf{o}_{<t} = (o_1, o_2, \dots, o_{t-1})$, where o_i is the i-th generated token. The policy π_θ becomes a conditional distribution over the next token o_t given the query and the previously generated tokens, i.e., $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$. In this sense, the tuple ($\mathbf{q}$, $\mathbf{o}_{<t}$) constitutes the state of the environment while the generated token o_t serves as the action taken by the agent. The process of appending the newly generated token to the sequence defines a deterministic environment transition.

With this formulation, RLVR typically relies on a fixed dataset $\mathcal{D} = \{(\mathbf{q}, y)\}_{1\dots N}$ consisting of a problem query $\mathbf{q}$ and some associated solution or verifier y. Coming from an RL perspective, each query $\mathbf{q}$ is an initial state, and the associated solution or verifier y is used to define a reward function $r(\mathbf{q}, \mathbf{o})$ that evaluates the quality of the generated token sequence. By the structure of this setup, the reward function is often sparse, only providing feedback at the end of a generated sequence. Most works further use a discount factor of $\gamma = 1$, causing many of the previously introduced concepts to simplify. For example, the advantage function of Equation 2.6, now denoted $A(o_t, \mathbf{q}, \mathbf{o}_{<t})$, is empirically estimated as $r(\mathbf{q}, \mathbf{o}) - V^{\pi_\theta}(\mathbf{q}, \mathbf{o}_{<t})$, where the final reward $r(\mathbf{q}, \mathbf{o})$ serves as a single-sample estimate of $Q^{\pi_\theta}(o_t, \mathbf{q}, \mathbf{o}_{<t})$.

Estimating this advantage requires a learned value function $V^{\pi_\theta}(\mathbf{q}, \mathbf{o}_{<t})$, which in LLM post-training is typically parameterized by another LLM-sized model, introducing significant training overhead and additional instability. However, the finite dataset combined with the non-physical nature of the environment enables efficient sampling of multiple trajectories from the same initial state. This property gives rise to the concept of group sampling, where multiple trajectories are sampled for each query in the dataset, and then used to estimate an advantage from the group performance. While many variants exist,

the original Group-Relative Policy Optimization (GRPO) (Shao et al., 2024) estimates the advantage for each token in the i-th group member as

$$\hat{A}^{(i)} = \frac{r^{(i)} - \bar{r}}{\sigma_r + \epsilon}$$

where $r^{(i)} = r(\mathbf{q}, \mathbf{o}^{(i)})$ is the reward for the i-th generated sequence, $\bar{r}$ is the mean reward of the group, and σ_r is the standard deviation of rewards within the group. In other words, the advantage becomes the sequence-level normalized difference between rewards in the group, which is applied to every token in the trajectory.

Beyond the challenge of advantage estimation, LLM post-training also involves optimizing policies in extremely high-dimensional action spaces, as the number of possible tokens can be in the tens of thousands. Similarly, the generated sequences can be several thousand tokens long, leading to a combinatorial explosion in the number of possible trajectories and corresponding states. The size of this search space makes it virtually impossible to solve tasks purely through exploration, which necessitates the use of a strong pre-trained model (Radford et al., 2019) as the policy's starting point. The high-dimensional action space in particular can easily lead to large differences between subsequent policies during training. As such, commonly employed on-policy objectives (Schulman et al., 2017) can quickly become unstable. Here, information-theoretic regularization can be used to enforce a notion of proximity between the current policy and the one used to gather the data, which can stabilize training and prevent catastrophic collapses in performance.

2.1.3. Variational Inference

The IL and RL frameworks introduced above can both be viewed as instances of distribution matching (Peters et al., 2010). IL aims to match the expert's state-action distribution, while RL seeks a policy whose induced distribution maximizes the expected return. VI provides a principled framework for such problems, where a tractable distribution q_θ is optimized to approximate an intractable target distribution p. Such an intractable distribution has an unknown normalization constant, meaning that it can not easily be sampled from. In the context of this thesis, these distributions are typically policies over actions, optimized either over states encountered during rollouts or jointly over all observed states in a demonstration dataset. Viewing policy optimization through this distributional lens connects it to established tools from information theory and variational methods, allowing us to leverage information-geometric principles for stable and efficient learning. In Chapter 3, we use VI to approximate an unknown, multimodal expert distribution from demonstrations with a Gaussian Mixture Model (GMM) policy. To formalize the comparison of probability distributions, we first introduce the concepts of entropy and divergence, which provide the mathematical language for the optimization frameworks developed throughout this thesis.

Entropy and Divergence. The Shannon entropy

$$H(p) = -\int p(x) \log p(x) \mathrm{d}x = \mathbb{E}_{p(x)} \left[-\log p(x) \right]$$

measures the uncertainty of a distribution $p(x)$. In the context of learning agents, an auxiliary entropy bonus (Schulman et al., 2017) or a maximum entropy objective (Haarnoja et al., 2018) is often used to encourage exploration and prevent the policy from prematurely collapsing into a single deterministic action. In discrete spaces, where the integral becomes a sum, the entropy is non-negative and can, using a $\log_2$ base, be interpreted as the expected number of bits required to encode a random variable drawn from the distribution. The entropy is a smooth measure that satisfies recursivity, meaning that the total uncertainty of a system can be decomposed into the sum of the uncertainties of its parts.

Building on the concept of entropy, the Kullback-Leibler (KL) divergence (Kullback et al., 1951) is a measure of how one probability distribution diverges from another. Given two distributions $p(x)$ and $q(x)$, the KL divergence from q to p is defined as

$$\mathrm{KL}\left(p \parallel q\right) = \int p(x) \log \frac{p(x)}{q(x)} \mathrm{d}x = \mathbb{E}_{p(x)} \left[\log \frac{p(x)}{q(x)} \right].$$

Intuitively, the KL divergence measures the additional information required to encode samples from p with the distribution q, or equivalently, the information gain when using p instead of q. The KL divergence is non-negative and is zero if and only if p and q are identical. Crucially, it is not symmetric, meaning that $\mathrm{KL}\left(p \parallel q\right) \neq \mathrm{KL}\left(q \parallel p\right)$ in general. This asymmetry leads to two different types of projections that we can use when approximating a target distribution with a parameterized and often learned distribution. These are the Moment-Projection (M-Projection) and the Information-Projection (I-Projection).

Moment- and Information-Projection. The M-Projection minimizes $\mathrm{KL}\left(p \parallel q_\theta\right)$ with respect to some parameters θ. It can be interpreted as finding a distribution q_θ that captures the moments of p, meaning that it is mass-covering and tends to spread the probability mass of q_θ to cover the entire support of p. In particular, it is zero-avoiding, meaning that it places non-zero probability for $q(x)$ on any region where $p(x) > 0$ due to the $\log(p(x)/q(x))$ term. A fundamental result in machine learning is the correspondence between optimizing the M-Projection and maximum likelihood estimation, where we find parameters θ that maximize the likelihood of the samples from p under the model q_θ, i.e.,

$$\begin{aligned}
\arg\min_\theta \mathrm{KL}\left(p \parallel q_\theta\right) &= \arg\min_\theta \int p(x) \log \frac{p(x)}{q_\theta(x)} \mathrm{d}x \\
&= \arg\min_\theta \left[\int p(x) \log p(x) \mathrm{d}x - \int p(x) \log q_\theta(x) \mathrm{d}x \right] \\
&= \arg\max_\theta H(p) + \mathbb{E}_{x\sim p}[\log q_\theta(x)] \\
&= \arg\max_\theta \mathbb{E}_{x\sim p}[\log q_\theta(x)],
\end{aligned}$$

where the entropy $H(p)$ of p is a constant and can be ignored for optimization. Similarly, for some distributions of the exponential family, such as Gaussian distributions $\mathcal{N}(\mu_\theta, \Sigma)$ with a fixed covariance Σ, it corresponds to minimizing the mean squared error between the parameterized mean μ_θ and samples from p, i.e.,

$$\begin{aligned}\arg\min_\theta \mathrm{KL}\left(p \parallel q_\theta\right) &= \arg\max_\theta \mathbb{E}_{x\sim p}[\log q_\theta(x)] \\ &= \arg\max_\theta \mathbb{E}_{x\sim p}\left[-\frac{1}{2}(x-\mu_\theta)^T\Sigma^{-1}(x-\mu_\theta)\right] + \text{const} \\ &= \arg\min_\theta \mathbb{E}_{x\sim p}\left[\|x-\mu_\theta\|^2_{\Sigma^{-1}}\right].\end{aligned}$$

The M-Projection works well for most regression and classification tasks, as it causes the distribution q_θ to accumulate as much probability mass of p as possible. Notably, BC as introduced in Section 2.1.1 implicitly performs an M-Projection, matching the expert's action distribution via supervised maximum likelihood estimation. However, the M-Projection's mass-covering property can be problematic when the target distribution p is multimodal or complex. In the context of learning versatile behaviors, an M-Projection would lead q_θ to average across all modes, potentially placing high probability mass in regions where p has none. Using the example of imitation learning from human behavior, averaging between expert modes may correspond to physically impossible and unsafe actions.

Swapping the terms in the KL divergence, the I-Projection instead minimizes $\mathrm{KL}\left(q_\theta \parallel p\right)$. Due to the $\log(q_\theta(x)/p(x))$ term, it is zero-forcing, meaning that it is heavily penalized for placing probability mass on regions where $p(x) \to 0$. For a limited-capacity model q_θ, the resulting projection is mode-seeking, meaning that the model q_θ tends to under-estimate the support of p and focus exclusively on the regions of highest probability. Figure 2.2 illustrates the difference between the M-Projection and the I-Projection using a simple example of a bimodal target distribution p and a unimodal Gaussian q_θ.

Evidence Lower Bound. While the I-Projection is the preferred objective for mode-seeking approximation, directly minimizing $\mathrm{KL}\left(q_\theta \parallel p\right)$ requires evaluating the target density p, which is often intractable. A key example arises in Bayesian inference, where we seek a posterior distribution $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$ over latent variables z given observed data x, but the marginal likelihood $p(x) = \int p(x,z)dz$ makes direct computation of this posterior impossible. VI formulates this challenge as an optimization problem where we seek a member of a tractable family of distributions $q_\theta(z|x)$ that minimizes the I-Projection to the true posterior, i.e., $\arg\min_{q_\theta} \mathrm{KL}\left(q_\theta(z|x) \parallel p(z|x)\right)$. We follow common notation and denote $q_\theta(z|x)$ as $q_\theta(z)$, as the conditioning on x is induced by the optimization objective.

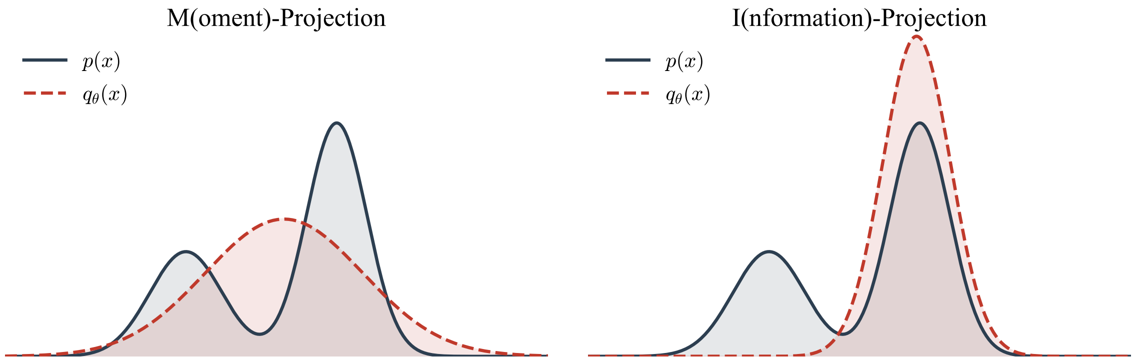


Figure 2.2.: Comparison between (M)oment- and (I)nformation-Projection. **(Left)** The M-Projection KL $(p \parallel q_\theta)$ minimizes the divergence from the target distribution p to the model q_θ, leading to a mass-covering solution that averages across modes. **(Right)** The I-Projection KL $(q_\theta \parallel p)$ minimizes the divergence from the model q_θ to the target distribution p, leading to a mode-seeking solution that focuses on one of the modes of p.

To derive a surrogate objective for this optimization, we decompose the log-evidence $\log p(x)$ by introducing the variational distribution $q_\theta(z)$ and applying the product rule of probability, i.e.,

$$
\begin{aligned}
\log p(x) &= \mathbb{E}_{z\sim q_\theta(z)}[\log p(x)] \\
&= \mathbb{E}_{z\sim q_\theta(z)}\left[\log \frac{p(x,z)}{p(z|x)}\right] \\
&= \mathbb{E}_{z\sim q_\theta(z)}\left[\log \frac{p(x,z)}{p(z|x)}\frac{q_\theta(z)}{q_\theta(z)}\right] \\
&= \mathbb{E}_{z\sim q_\theta(z)}\left[\log \frac{p(x,z)}{q_\theta(z)}\frac{q_\theta(z)}{p(z|x)}\right] \\
&= \mathbb{E}_{z\sim q_\theta(z)}\left[\log \frac{p(x,z)}{q_\theta(z)}\right] + \mathbb{E}_{z\sim q_\theta(z)}\left[\log \frac{q_\theta(z)}{p(z|x)}\right] \\
&= \underbrace{\mathbb{E}_{z\sim q_\theta(z)}\left[\log p(x,z)\right] + H(q_\theta(z))}_{\text{ELBO}} + \underbrace{\mathrm{KL}\left(q_\theta(z) \parallel p(z|x)\right)}_{\geq 0}.
\end{aligned}
\tag{2.7}
$$

Since the KL divergence is always non-negative, the first term on the right-hand side serves as a lower bound on the log-evidence, hence the name Evidence Lower Bound (ELBO). Crucially, because the log-evidence $\log p(x)$ is determined solely by the model and the data, it remains constant with respect to the variational parameters θ. This constant relationship implies a zero-sum trade-off between the two terms, meaning that any increase in the ELBO must be met by a corresponding decrease in the KL divergence. Consequently, maximizing the ELBO is mathematically equivalent to minimizing the I-Projection to the posterior.

Expectation Maximization. The ELBO decomposition of Equation 2.7 directly gives rise to the Expectation Maximization (EM) algorithm, which uses it to find maximum likelihood estimates of model parameters ϕ in latent variable models. EM can be seen as a special case of VI where the variational distribution $q_\theta(z)$ is set to the exact posterior $p(z|x, \hat{\phi})$ at each iteration. A typical use case for EM is the training of a Gaussian Mixture Models (GMMs) $p(x, z|\phi)$, which is defined as a sum $p(x|\phi) = \sum_z p(z|\phi)p(x|z, \phi)$ over multivariate Gaussian components $p(x|z, \phi) = \mathcal{N}(\mu_z, \Sigma_z)$ weighted by a categorical distribution $p(z|\phi) \in \text{Cat}(\boldsymbol{\pi})$. The parameters ϕ include the means μ_z, covariances Σ_z, and mixing coefficients of the Gaussian components.

EM alternates between the E(xpectation)-Step and the M(aximization)-Step. The E-Step fixes the current parameters $\hat{\phi} \leftarrow \phi$ and updates θ to minimize KL $\left(q_\theta(z) \parallel p(z|x, \hat{\phi})\right)$, which tightens the ELBO. In the classical setting, where q_θ is unconstrained, the I-Projection term becomes zero, making the ELBO equal to the log-evidence. For models where the exact posterior is intractable, existing extensions instead rely on partial updates or variational approximations (Neal et al., 1998). For GMMs, this step corresponds to calculating the responsibilities of the mixture components, which can be expressed in closed form using Bayes' theorem, i.e.,

$$q_\theta(z) = \frac{p(x|z, \hat{\phi})p(z|\hat{\phi})}{\sum_{z'} p(x|z', \hat{\phi})p(z'|\hat{\phi})}$$

The M-Step then fixes $q_\theta(z)$ and maximizes the ELBO with respect to ϕ, i.e.,

$$\phi = \arg\max_{\phi'} \mathbb{E}_{z \sim q_\theta(z)}[\log p(x, z|\phi')],$$

where the $H(q_\theta(z))$ term is constant with respect to ϕ' and can be ignored for optimization. By alternating between these steps, EM ensures that the marginal likelihood $p(x|\phi)$ is non-decreasing at each iteration, eventually converging to a local maximum of the likelihood function. EM can be applied to IL, providing, e.g., a simple multimodal instantiation of behavioral cloning with a GMM policy.

Expected Information Maximization. While EM provides closed-form updates for tractable models like GMMs, it inherently performs an M-Projection. As discussed above, M-Projections tend to average over modes, potentially failing to capture the full structure of multimodal targets. In the context of learning from versatile human behavior, this averaging can produce physically impossible or unsafe actions that lie between distinct expert strategies. Expected Information Maximization (EIM) (Becker et al., 2020) addresses this limitation by replacing the M-Projection inherent in EM's parameter update with an I-Projection, providing a mode-seeking alternative that operates directly on demonstration data. In the following, we drop the explicit parameter subscripts to avoid notational clutter.

Like EM, EIM optimizes a GMM policy $q(x) = \sum_z q(z)q(x|z)$ through an alternating scheme, updating each mixture component independently. However, instead of closed-form E-Step updates, EIM uses trust-region-constrained policy search (Peters et al., 2010; Abdolmaleki et al., 2015) to optimize the per-component objectives. The key challenge in applying these methods is that the expert distribution $p(x)$ is unknown and only available through a finite dataset of demonstrations. To obtain a learning signal, EIM trains a binary classifier to estimate the log-density ratio $\boldsymbol{\psi}(x) = \log \frac{p(x)}{\hat{q}(x)}$ between the data distribution and the current model $\hat{q}(x)$ from the previous iteration. Concretely, training a binary classifier using a simple cross-entropy loss to distinguish between samples from the dataset and samples from the current model $\hat{q}(x)$ yields an optimal discriminator (Sugiyama et al., 2012)

$$D^*(x) = \frac{p(x)}{p(x) + \hat{q}(x)}. \tag{2.8}$$

Assuming the discriminator is parameterized by logits $\boldsymbol{\psi}(x)$ such that $D(x) = \sigma(\boldsymbol{\psi}(x))$, where $\sigma(\cdot)$ is the sigmoid function, the logits of the optimal discriminator estimate the log-density ratio, i.e.,

$$\boldsymbol{\psi}^*(x) = \sigma^{-1}(D^*(x)) = \log \frac{D^*(x)}{1 - D^*(x)} = \log \frac{\frac{p(x)}{p(x)+\hat{q}(x)}}{\frac{\hat{q}(x)}{p(x)+\hat{q}(x)}} = \log \frac{p(x)}{\hat{q}(x)}. \tag{2.9}$$

The trained classifier can then be used as a surrogate reward $R(x) = \boldsymbol{\psi}(x) \approx \log p(x) - \log \hat{q}(x)$ for the trust-region policy search problem that updates each mixture component. Crucially, the utilized policy search methods enforce a KL bound between subsequent iterations, updating each component within the intrinsic geometry of its distribution rather than in raw parameter space. This constraint allows EIM to optimize GMM policies to match versatile expert behavior from demonstrations while maintaining stable convergence. The formal variational objective underlying this reward, and its derivation from the I-Projection, are developed in the following.

Variational Inference by Policy Search. EIM's reward $R(x) = \log p(x) - \log \hat{q}(x)$ can be understood as a special case of a more general variational framework. Variational Inference by Policy Search (VIPS) (Arenz et al., 2018; Arenz et al., 2020) derives this reward structure from first principles by decomposing the I-Projection $\mathrm{KL}(q(x) \parallel p(x))$ for GMM policies. While EIM requires only samples from the target, VIPS addresses the more general setting where the target is specified through an unnormalized density $\tilde{p}(x)$ with $p(x) = \tilde{p}(x)/Z$ and intractable normalization constant $Z = \int \tilde{p}(x)\mathrm{d}x$.

To enable independent optimization of the GMM components, VIPS introduces an auxiliary distribution $\hat{q}(x, z) = \hat{q}(z)\hat{q}(x|z)$ representing the model from the previous iteration. Using the identity $\log q(x) = \log q(x, z) - \log q(z|x)$, the variational objective can be decomposed as (Arenz et al., 2018; Arenz et al., 2020; Becker et al., 2020)

$$
\begin{aligned}
\mathrm{KL}(q(x) \parallel p(x)) &= \mathbb{E}_{q(x)}\left[\log \frac{q(x)}{\tilde{p}(x)}\right] + \log Z \\
&= \mathbb{E}_{q(x,z)}\left[\log \frac{q(x,z)}{\tilde{p}(x)q(z|x)}\right] + \log Z \\
&= \mathbb{E}_{q(x,z)}\left[\log \frac{q(x,z)}{\tilde{p}(x)\hat{q}(z|x)} \cdot \frac{\hat{q}(z|x)}{q(z|x)}\right] + \log Z \\
&= \underbrace{\mathbb{E}_{q(x,z)}\left[\log \frac{q(z)q(x|z)}{\tilde{p}(x)\hat{q}(z|x)}\right]}_{\mathcal{L}_{\text{VIPS}}} - \underbrace{\mathbb{E}_{q(x)}\left[\mathrm{KL}(q(z|x) \parallel \hat{q}(z|x))\right]}_{\geq 0} + \underbrace{\log Z}_{const.}.
\end{aligned}
\tag{2.10}
$$

We can drop the constant $\log Z$, leaving the VIPS objective and a non-positive KL term. The latter acts as a tightening bound when q stays close to $\hat{q}$. We can now further manipulate $\mathcal{L}_{\text{VIPS}}$ by substituting $\hat{q}(z|x) = \frac{\hat{q}(x|z)\hat{q}(z)}{\hat{q}(x)}$ into the numerator, yielding

$$
\begin{aligned}
\mathcal{L}_{\text{VIPS}} &= \mathbb{E}_{q(x,z)}\left[\log \frac{q(z)q(x|z)\hat{q}(x)}{\tilde{p}(x)\hat{q}(x|z)\hat{q}(z)}\right] \\
&= \mathbb{E}_{q(x)}\left[\log \frac{\hat{q}(x)}{\tilde{p}(x)}\right] + \mathbb{E}_{q(z)}\left[\log \frac{q(z)}{\hat{q}(z)}\right] + \mathbb{E}_{q(x,z)}\left[\log \frac{q(x|z)}{\hat{q}(x|z)}\right] \\
&= \mathbb{E}_{q(x)}\left[\log \frac{\hat{q}(x)}{\tilde{p}(x)}\right] + \mathrm{KL}(q(z) \parallel \hat{q}(z)) + \mathbb{E}_{q(z)}[\mathrm{KL}(q(x|z) \parallel \hat{q}(x|z))].
\end{aligned}
$$

The decomposition in Equation 2.10 allows independent optimization of the mixture components, as the coupling term $\log q(z|x)$ is replaced by the frozen model $\hat{q}(z|x)$. After tightening the KL terms by setting $\hat{q} \leftarrow q$, the per-component optimization problem for $q(x|z)$ becomes

$$
\min_{q(x|z)} \mathbb{E}_{q(x|z)}\left[\log \frac{\hat{q}(x)}{\tilde{p}(x)}\right] + \mathrm{KL}(q(x|z) \parallel \hat{q}(x|z)),
$$

which can be solved as a trust-region policy search problem (Peters et al., 2010; Abdolmaleki et al., 2015) with reward $R(x) = \log \tilde{p}(x) - \log \hat{q}(x)$. This reward recovers the log density ratio used by EIM in Equation 2.9, instantiating EIM's classifier-based reward as a principled variational objective. The original VIPS derivation (Arenz et al., 2018) is slightly different, yielding a reward $R(x) = \log \tilde{p}(x) + \log \hat{q}(z|x)$ and an additional entropy term instead of a KL term. Both derivations are equivalent, and the difference can be accounted for by adapting the Lagrangian parameters of the dual function of the policy search problem (Becker et al., 2020). The categorical distribution $q(z)$ can be optimized using a similar objective. Both EIM and VIPS mirror EM in their alternating optimization scheme, where the auxiliary distribution $\hat{q}$ is updated to the current model q after each iteration.

However, unlike EM, which relies on closed-form updates of the E-Step, they use trust-region policy search to optimize a per-component I-Projection objective. In particular, stable optimization during the policy search is ensured by bounding the KL divergence between subsequent iterations. Formalizing and generalizing these KL constraints leads to the information-geometric framework introduced in the following section.

2.2. Information-Geometric Inductive Biases

The trust-region constraints encountered in the previous section are instances of a broader class of methods treating probability distributions as points on a statistical manifold. These methods then use the intrinsic geometry of this manifold to guide optimization. Information geometry (Amari, 2016) provides the mathematical framework to formalize this idea, equipping the space of distributions with a natural metric that measures distances based on informational content rather than raw parameter values. In the context of policy optimization, this geometric perspective gives rise to trust-region methods that can guarantee stable, monotonic policy improvements by constraining each update to a local neighborhood on the manifold (Kakade et al., 2002a; Peters et al., 2010; Schulman et al., 2015a). These information-geometric inductive biases are central to several contributions of this thesis. The multi-agent mesh refinement of Chapter 4 relies on Proximal Policy Optimization (PPO) (Schulman et al., 2017), a first-order approximation of trust regions, for stable multi-agent training. This approximation is challenged and improved in Chapter 6, which replaces it with trust region projections for large discrete action spaces and LLM post-training. Finally, the inner-loop optimization of Chapter 3 relies on EIM, which itself uses trust-region policy search to update each mixture component using a learned density ratio as the reward.

Fisher Information Matrix. The central object in information geometry is the Fisher Information Matrix $F(\theta)$ (Fisher, 1922), which serves as the Riemannian metric tensor of the statistical manifold. It measures distances between nearby distributions based on their informational content rather than their raw parameter values, and is defined as

$$F(\theta) = \mathbb{E}_{x\sim p_\theta}\left[\nabla_\theta \log p_\theta(x) \nabla_\theta \log p_\theta(x)^T\right].$$

Intuitively, the Fisher Information Matrix captures how sensitive a distribution is to small changes in its parameters. For example, a fixed shift in the mean of a Gaussian distribution represents a much larger change in information if the variance is small than if the variance is large. The Fisher Information Matrix accounts for this by inducing a higher curvature in sensitive regions of the parameter space, which effectively scales down optimization steps to prevent small parameter updates from causing disproportionately large changes in behavior. A key connection to the divergence measures introduced in the previous section is that the Fisher Information Matrix arises as the second-order Taylor expansion of the KL divergence around a reference distribution, i.e.,

$$\mathrm{KL}\left(p_\theta \parallel p_{\theta+\delta\theta}\right) \approx \frac{1}{2}\delta\theta^T F(\theta)\delta\theta,$$

where the zero- and first-order terms vanish. This relationship means that the Fisher metric provides a suitable local approximation of the KL divergence between nearby distributions, making it the natural tool for constraining policy updates. The Fisher metric also gives rise to the Natural Gradient $\tilde{\nabla} f(\theta) = F(\theta)^{-1} \nabla f(\theta)$, which ensures optimization steps are invariant to the specific parameterization of the policy (Kakade, 2001).

Trust Region Policy Optimization. The concept of a trust region stems from the realization that first-order optimization steps are only reliable within a local neighborhood of the current parameters. When optimizing a policy, a large step in the wrong direction can move the agent into regions of the parameter space that produce catastrophic actions, potentially leading to a collapse in performance and policy divergence. Trust region methods mitigate this risk by defining a maximum allowable change to the policy at each iteration. Instead of taking a step of fixed size in the gradient direction, these methods seek the best possible update within a safe distance of the current policy.

The I-Projection's mode-seeking behavior is particularly suitable for this trust region constraint. By ensuring $\mathrm{KL}\left(\pi_\theta \parallel \pi_{\text{old}}\right)$ remains small, we guarantee that actions sampled from the new policy are likely under the old policy. This approach provides a theoretical foundation for stable, monotonic policy improvements (Kakade et al., 2002a; Peters et al., 2010; Schulman et al., 2015a).

In practice, the on-policy policy gradient derived in Section 2.1.2 requires collecting fresh trajectories for every gradient step, making it expensive and time-consuming. To enable multiple updates from a single batch of data, we can employ importance sampling by introducing a policy ratio $\omega_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$, which re-weights samples collected under π_{old} to estimate the gradient of the current policy π_θ. This ratio replaces the log-probability in the policy gradient with a likelihood ratio, yielding the surrogate objective $\mathbb{E}_{(s,a)\sim\rho^{\pi_{\text{old}}}}[\omega_t(\theta) A^{\pi_{\text{old}}}(s,a)]$. As the current policy diverges from the data-collecting policy, this ratio can become very large or very small, causing individual samples to dominate the update and leading to training instability. Trust region methods address this by constraining how far the new policy may deviate from the old one.

Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a) formalizes this idea by optimizing the surrogate objective subject to an explicit KL constraint:

$$\begin{aligned}\max_{\pi_\theta} \mathcal{J}_{\text{trpo}}(\pi_\theta) &= \mathbb{E}_{(s,a)\sim\rho^{\pi_{\text{old}}}}\left[\omega_t(\theta) A^{\pi_{\text{old}}}(s,a)\right] \\ \text{s.t. } &\mathbb{E}_{(s,a)\sim\rho^{\pi_{\text{old}}}}\left[\mathrm{KL}\left(\pi_{\text{old}}(\cdot|s) \parallel \pi_\theta(\cdot|s)\right)\right] \leq \delta,\end{aligned}$$

where the KL constraint is approximated using the Fisher Information Matrix. However, TRPO is computationally expensive, especially for modern deep neural network policies, because it requires calculating second-order derivatives and solving the constrained optimization problem via the conjugate gradient method. Other trust region methods (Abdolmaleki et al., 2018; Song et al., 2020) relax the state-level constraint and instead rely on regularizing the expected KL between subsequent policies.

Proximal Policy Optimization. Rather than solving TRPO's constrained optimization problem, Proximal Policy Optimization (PPO) (Schulman et al., 2017) replaces the explicit trust region constraint with a clipping-based heuristic on the policy ratio. Instead of bounding the KL divergence, PPO clips the policy ratio $\omega_t(\theta)$ to the interval $[1-\epsilon, 1+\epsilon]$, yielding the objective

$$\mathcal{J}_{\text{ppo}}(\pi_\theta) = \mathbb{E}_{(s,a)\sim\rho^{\pi_{\text{old}}}}\left[\min(\omega_t(\theta)\hat{A}(s,a), \text{clip}(\omega_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}(s_t,a_t))\right] \quad (2.11)$$

to prevent this divergence. The advantage $\hat{A}(s_t, a_t)$ is typically estimated using Generalized Advantage Estimation (Schulman et al., 2015b) with a learned value function, and the clipping parameter ϵ controls how much the policy is allowed to change at each update step. Due to its simplicity and strong empirical performance, PPO has become arguably the most widely used on-policy optimization algorithm in practice, serving as the default choice in domains ranging from robotics (Andrychowicz et al., 2021) and game-playing (Berner et al., 2019) to LLM post-training (Ouyang et al., 2022). However, the clipping mechanism is merely a heuristic approximation of the trust region. It does not enforce any explicit bound on the KL divergence between subsequent policies, and gradients vanish for samples whose policy ratio falls outside the clipping range, discarding potentially useful learning signal.

Trust Region Projections. To address PPO's shortcomings, Trust Region Projection Layers (TRPL) (Otto et al., 2021) takes a fundamentally different approach by integrating trust regions directly into the model architecture as a differentiable projection layer. In this formulation, an unconstrained neural network outputs the parameters of a policy $\tilde{\pi}_\theta$, which is subsequently projected back onto the trust region defined by the reference policy π_{old} by solving

$$\pi_\theta = \arg\min_\pi \text{KL}\left(\pi(\cdot|s) \,\|\, \tilde{\pi}_\theta(\cdot|s)\right) \quad \text{s.t. KL}\left(\pi(\cdot|s) \,\|\, \pi_{\text{old}}(\cdot|s)\right) \leq \delta$$

for all $s \in \mathcal{S}$. In practice, the constraint can only be achieved for states that are visited by the reference policy π_{old}, which is sufficient for ensuring stable policy updates. This projection is implemented as the final layer of the policy network, ensuring that the trust region constraint is enforced analytically for every sampled action. Compared to the iterative second-order optimization of TRPO, this approach enforces state-level constraints within a standard first-order optimization framework. However, the trust region projection requires solving a convex optimization problem during each forward pass and backpropagating through the projection operator using implicit differentiation (Dontchev et al., 2009).

In principle, each new optimization step introduces a new reference policy π_{old}, which would require a new projection layer to enforce the trust region constraint relative to that specific iteration. This process would lead to a growing stack of projection layers, which is computationally expensive and impractical. The authors propose to instead use a single projection layer, which is coupled with a regression loss between the unconstrained policy $\tilde{\pi}_\theta$ and the projected policy π_θ. TRPL has so far only been applied to Gaussian distributions and continuous domains.

2.3. Geometric Inductive Biases

The previous section formalized geometry in the space of probability distributions, using the curvature of the statistical manifold to guide policy optimization. We now shift to geometry in the physical world, where the structure of the data itself provides powerful inductive biases for learning. For instance, in robotic manipulation, the relative positions and orientations of objects often determine the task's success, while in physical simulation, the underlying physics is largely governed by spatially local interactions. This section introduces the geometric inductive biases that we leverage in Chapter 3, Chapter 4 and Chapter 5 to enable representation of and learning over complex, irregular domains.

In particular, we start by introducing physical simulation using the Finite Element Method (FEM), which is a cornerstone of many engineering disciplines and serves as the application domain for the adaptive meshing algorithms developed in Chapter 4 and Chapter 5. Since meshes in the FEM are naturally represented as graphs, we then introduce the principles of GDL and Graph Neural Networks (GNNs), which provide the tools to model and learn over such irregular, graph-structured data. The Euclidean invariances formalized here are also central to the behavioral descriptors introduced in Chapter 3.

2.3.1. Finite Element Method

Physical simulation serves as the cornerstone of modern numerical engineering, enabling the prediction and analysis of complex systems through the solution of Partial Differential Equations (PDEs). The Finite Element Method (FEM) has emerged as a popular numerical framework due to its ability to handle arbitrary, irregular geometries as well as complex material properties and boundary conditions. At its core, the FEM discretizes a continuous domain into a mesh of elements and defines local approximation functions on them, transforming continuous operators into solvable algebraic equations. While not the focus of this thesis, we briefly review the underlying FEM basics to provide the necessary context for the adaptive meshing algorithms developed in Chapter 4 and Chapter 5. The following review follows the standard treatments found in Fish et al. (2007), Larson et al. (2013) and Brenner et al. (2008).

Strong and Weak Formulations. Consider the Poisson equation as a representative example problem that appears in applications ranging from heat conduction to electrostatics. The Poisson equation is a prototypical elliptic PDE, a class of equations that describes steady-state processes where local changes have an instantaneous, global effect on the domain. The strong formulation of the Poisson equation is given by

$$-\Delta u = f \text{ in } \Omega, \quad u = 0 \text{ on } \partial\Omega,$$

where f is a source term and $u = u(\mathbf{x})$ is a space-dependent solution representing the unknown physical equilibrium state. The condition $u = 0$ on $\partial\Omega$ is a Dirichlet boundary condition, which physically represents a fixed state, such as a fixed temperature or grounded electrical potential at the boundary of the domain. Depending on the source term f and the domain geometry, the solution's smoothness often varies significantly across different regions. Even for a simple example like the Poisson problem, there are usually no analytical solutions for such a strong formulation, since the domain geometry, source term f and boundary conditions may not allow for integration in closed form. Instead, the FEM considers the weak formulation (Larson et al., 2013). By multiplying the strong form with a test function $v(\mathbf{x}) \in V$ and integrating by parts, we yield the weak form $\Phi(u, v) = L(v)$. For our Poisson example, this results in the bilinear form $\Phi(u, v)$ and the linear form $L(v)$, defined as

$$\Phi(u, v) = \int_{\Omega} \nabla u \cdot \nabla v \, \mathrm{d}\mathbf{x} \quad \text{and} \quad L(v) = \int_{\Omega} f v \, \mathrm{d}\mathbf{x}.$$

This formulation is the standard starting point for numerical discretization because it is mathematically more well behaved than the strong form. For instance, the Lax-Milgram theorem provides clear conditions under which a unique solution in the Hilbert space V is guaranteed to exist (Brenner et al., 2008). Additionally, it naturally incorporates natural boundary conditions, such as Neumann conditions, which appear directly as terms within the bilinear form during integration by parts. This contrasts with essential boundary conditions, like the Dirichlet condition $u = 0$ in the example above, which are instead enforced by restricting the function space V. Further, the weak form reduces the differentiability requirements on the solution u by redistributing the derivative operators to the test function v. This property is crucial for practical numerical methods as it allows the solution to be approximated by simple piecewise functions not necessarily sufficiently smooth to satisfy the strong form.

Galerkin Finite Element Method. In the Galerkin FEM, the infinite-dimensional solution $u(\mathbf{x})$ is approximated within a finite-dimensional subspace $V_h \subset V$. The Galerkin approach specifically chooses both the trial and test functions from this same subspace. This choice leads to the property of Galerkin orthogonality, where the error of the approximation is orthogonal to the subspace V_h. Intuitively, the Galerkin FEM therefore seeks a solution that satisfies the PDE in an average sense against all possible variations represented by the discrete test functions. Testing against all $v \in V_h$ means that the residual of the equation must be orthogonal to the discrete function space, ensuring the approximation is as close to the solution of the continuous problem as the space V_h allows. The solution is expressed as a weighted sum of basis functions $\boldsymbol{\phi}_v(\mathbf{x}) \in V_h$ as

$$u(\mathbf{x}) \approx \sum_{v=1}^{N_\varphi} u_v \boldsymbol{\phi}_v(\mathbf{x}).$$

By choosing the test functions v from the same basis as the trial functions u, the problem reduces to a linear system of equations

$$\mathbf{A}\mathbf{u} = \mathbf{b}.$$

Here, $\mathbf{A}_{ij} = \Phi(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i)$ represents the stiffness matrix and $\mathbf{b}_i = L(\boldsymbol{\phi}_i)$ the load vector. The coefficients $\mathbf{u} = \left[u_1, \dots, u_{N_\varphi}\right]^\top$ represent the discrete degrees of freedom.

For time-dependent parabolic PDEs, such as the heat equation, we additionally require a discretization in time, for example via Euler schemes. In these cases, the time derivative is typically replaced by a finite difference approximation, transforming the problem of temporal evolution into a sequence of stationary problems. The resulting linear systems causally depend on each other and must be solved iteratively at each time step to advance the solution in time.

Meshes. Discretizing the weak form into a computable linear system requires partitioning the domain into finite elements and defining appropriate basis functions on them. A mesh M is such a partition of the domain Ω into a finite set of non-overlapping elementary geometric shapes, called elements, M_i, such that $\bar{\Omega} = \bigcup_{M_i \in M} M_i$. Here, $\bar{\Omega}$ denotes the closure of the domain, which includes its boundary. While many different element types exist, this thesis focuses on simplicial elements, i.e., triangles in 2D and tetrahedra in 3D due to their simplicity and flexibility in handling complex geometries. As such, the meshes discussed in this work are triangulations of the underlying domain. In the context of the FEM, the mesh provides the topological connectivity and geometric information necessary to evaluate the integrals in the weak form. For our learning agents, it acts as the graph, either per element or per node, on which the Message Passing Network (MPN) architecture operates.

Basis Functions. To construct the finite-dimensional subspace V_h for Galerkin FEM, we define a set of basis functions $\{\boldsymbol{\phi}_i\}_{i=1}^{N_\varphi}$ with local support tied to the nodes of the mesh. Typically, basis functions are Lagrange polynomials $\mathbb{P}_k$ of degree k, meaning that they take a value of one at their associated node and zero at all other nodes in the element. This property allows them to interpolate the solution across the elements with which they are associated. For linear elements, i.e., $k = 1$, these nodes directly correspond to the vertices of the mesh, and the basis function $\boldsymbol{\phi}_v$ is non-zero only on the elements that share the vertex v. For $k > 1$, nodes also include points along the edges or within the interior of the elements. Basis functions are constructed to be C^0-continuous across element boundaries but are only piecewise smooth, as required by the weak formulation. The local support property is critical because it ensures that the resulting stiffness matrix $\mathbf{A}$ is sparse, allowing for efficient linear solvers. We will mostly focus on linear basis functions, i.e., $k = 1$, where the solution on the mesh is approximated by a piecewise linear function, i.e., linearly interpolated at the vertices of the mesh.

Adaptive Mesh Refinement. The primary motivation for Adaptive Mesh Refinement (AMR), and adaptive meshing in general, is numerical efficiency. The goal is to obtain a solution with the highest possible accuracy for a given computational cost. This goal is typically achieved by seeking an equidistribution of the approximation error, where the mesh is selectively densified in regions of high solution complexity while remaining coarse elsewhere. We differentiate between three main AMR strategies. These are h-refinement, which refines the mesh by adding more elements, p-refinement, which increases the order of the basis functions for some elements, and r-refinement, which redistributes or moves existing mesh points without changing the number of elements. In this thesis, we focus on h-refinement, i.e., on refinement via element sub-division, which is the most common approach in practice and allows us to leverage the same, simple basis functions across different mesh resolutions.

The h-refinement process is typically iterative. In each step, a marking strategy identifies a subset of elements for refinement based on local features, such as error indicators for the underlying simulation. These elements are then refined into multiple smaller elements. To ensure the stability and accuracy of the subsequent FEM solution, the refinement must ensure certain aspect ratios of the elements as well as conformity of the mesh. A mesh is considered conforming if the intersection of any two elements is either empty or a shared vertex, edge, or face, which ensures that the mesh does not contain any hanging nodes. Both conditions may require additional elements to be refined, even if they were not originally marked for refinement. Due to the iterative nature of this process and the direct dependency of a refined element on its parent, the refinement hierarchy can be represented as a forest. Here, each root corresponds to an initial element of the coarse base mesh, with children representing successive refinements. This tree-like structure also allows for de-refinement, or coarsening, where certain elements are merged back together, which is particularly useful for time-dependent problems where the regions of interest may change over time.

Adaptive Mesh Generation. In contrast to the iterative sub-division of AMR, Adaptive Mesh Generation (AMG) involves the synthesis of a new mesh from scratch based on specific underlying criteria. While standard AMR is constrained by the topology of the initial base mesh, AMG provides the flexibility to more closely align the mesh to the geometry of the problem. This flexibility is helpful when the initial mesh is too coarse to capture complex domain features, a scenario that would require complex projection steps in AMR. For our purposes, the generation process is guided by a sizing field, which is a continuous function that defines the desired element edge length or mesh density at each point on the domain. AMG effectively generates a mesh that adheres to this sizing field, allowing us to provide a distribution of mesh elements that efficiently captures the solution's complexity, similar to the equidistribution goal of AMR. Although the remeshing process is often computationally more expensive than local refinement, it can yield significantly better meshes, as there is no hierarchical dependency on the initial mesh.

The meshes produced by both strategies are inherently graph-structured, with nodes, edges, and elements forming irregular topologies that vary across problem instances. Designing learning algorithms that can effectively operate on such structures requires architectural choices that respect the underlying symmetries of these graphs, which we formalize in the following.

2.3.2. Symmetries and Inductive Biases

Symmetries express transformations that leave certain properties of an entity unchanged, and encoding them as inductive biases into neural architectures can dramatically reduce the effective dimensionality of the learning problem. In this work, we leverage Euclidean invariance for the geometric behavioral descriptors of Chapter 3, and permutation and Euclidean equivariance for the graph-based mesh representations of Chapter 4 and Chapter 5.

Universal Approximation and the Curse of Dimensionality. Modern neural networks are universal function approximators for continuous functions on compact sets (Cybenko, 1989; Hornik, 1991). Loosely speaking, this means that they can approximate the target function to arbitrary precision, given sufficient capacity and data. While this is an uplifting theoretical result, modern applications of neural architectures are required to process high-dimensional data, be it signals over time, images, or, as in our case, unstructured graphs. When disregarding its structure, a naive implementation of a neural network requires an exponential number of samples to learn a function over this data, which is known as the curse of dimensionality. One of the fundamental challenges in GDL therefore lies in designing architectures that learn effectively in such high-dimensional spaces by using the inherent structure of the data to reduce the effective dimensionality of the learning problem.

Symmetries and Groups. A symmetry of an entity is a transformation that leaves certain properties of the entity unchanged. Mathematically, the collection of all symmetries in a system forms a group. Formally, these transformations are organized into a group G, a set equipped with a binary composition operator that satisfies closure, associativity, identity, and inversion.

In the context of this thesis, we are interested in symmetries of the data and the learning problem, which can be leveraged to design architectures that are invariant or equivariant to these symmetries. Simple examples include discrete translation symmetries in images, which can be leveraged by, e.g., Convolutional Neural Network (CNN) (LeCun et al., 2002). We are generally interested in geometric symmetries, such as Euclidean symmetries, which include translations, rotations and reflections in space, as well as permutations of the nodes in a graph. Such symmetries are prevalent in real-world robotic tasks, where the position and rotation of objects in the environment can vary, as well as in the case of mesh-based physical simulations, where the coordinate system is arbitrary and the ordering of nodes in the graph is irrelevant.

Invariance and Equivariance. To formalize functions that respect these symmetries, we consider a group G and its action on the data space. Let $f : \mathcal{X} \to \mathcal{Y}$ be a function, such as a neural network layer, and let ρ_G and ρ'_G be the representations of the group G acting on the input space $\mathcal{X}$ and output space $\mathcal{Y}$, respectively. A function is said to be

G-equivariant if it commutes with the group action, meaning that transforming the input results in a corresponding transformation of the output, i.e.,

$$f(\rho_G(g)x) = \rho'_G(g)f(x)$$

for all group elements g and inputs x. Similarly, a function is G-invariant if the output space remains unaffected by the group action, i.e.,

$$f(\rho_G(g)x) = f(x) \quad \forall g \in G, x \in \mathcal{X}$$

for all group elements g and inputs x.

By constraining the hypothesis space of our models to functions that satisfy these symmetries, we explicitly bake inductive biases directly into the architecture. Coming back to our example of images, a CNN is designed to be equivariant to discrete translations, meaning that if the input image is shifted, the output feature maps will shift accordingly. In the case of graph-based data, GNNs layers are designed to be equivariant to the permutation group S_n acting on the node indices, ensuring that the model's predictions do not depend on the arbitrary ordering of nodes in the graph.

2.3.3. Graph Neural Networks

Graph Neural Networks (GNNs) (Scarselli et al., 2009) are arguably the most natural architecture for processing graph-structured data, as they operate directly on the graph's nodes and edges. By employing permutation-invariant aggregations over the neighborhood of each node, GNNs ensure a node-wise equivariant representation that effectively captures the relational structure of the underlying data. In this thesis, we utilize GNNs in Chapter 4 and Chapter 5 to process mesh-based data, treating mesh elements and vertices as nodes and their connectivity as edges.

Graphs and Sets. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ that connect pairs of nodes. Graphs are therefore one of the most fundamental data structures, finding use in applications ranging from modeling social networks and molecular structures to conducting physical simulations. Since nodes in a graph are typically unordered, operations on graphs should also not depend on the node order, which is where the aforementioned permutation equivariance comes into play.

To formalize this, we must first consider that graphs may also include node and edge features that provide additional information about the represented entities and their relation. Node features can be represented as a matrix $\mathbf{X}_{\mathcal{V}} \in \mathbb{R}^{|\mathcal{V}| \times d}$, where d is the dimensionality of the features, and each row corresponds to the feature vector of a node. The same concept applies to edge features. The graph structure itself is captured by the adjacency matrix $\mathbf{A}$, which encodes the connectivity between nodes as

$$a_{uv} = \begin{cases} 1 & (u, v) \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases}$$

From these definitions, we can define graph-wise permutation invariant and node-wise permutation equivariant functions $f(\mathbf{X}_\mathcal{V}, \mathbf{A})$ as

$$f(\mathbf{P}\mathbf{X}_\mathcal{V}, \mathbf{P}\mathbf{A}\mathbf{P}^T) = f(\mathbf{X}_\mathcal{V}, \mathbf{A})$$

and

$$f(\mathbf{P}\mathbf{X}_\mathcal{V}, \mathbf{P}\mathbf{A}\mathbf{P}^T) = \mathbf{P}f(\mathbf{X}_\mathcal{V}, \mathbf{A}),$$

respectively. Here, $\mathbf{P}$ is a permutation matrix that reorders the nodes in the graph, i.e., that contains exactly one entry of 1 in each row and column and 0s elsewhere. Consequently, f must map to a graph-level representation in the former case, and to a node-level representation in the latter.

Neighborhood and Receptive Fields. The local structure of a graph is defined by the neighborhood of its nodes. For a node $u \in \mathcal{V}$, the set of its adjacent nodes is denoted as $N(u) = \{v \in \mathcal{V} : (u, v) \in \mathcal{E}\}$. Most GNN architectures deliberately operate on this local neighborhood, allowing them to capture the relational structure of the graph while maintaining computational efficiency. Each GNN layer allows information to be exchanged between neighboring nodes. In these architectures, a single layer typically aggregates information from the direct 1-hop neighborhood, while stacking k layers allows a node to incorporate from k-hop nodes, i.e., from nodes whose shortest path to the reference node is of length k. This mechanism allows the network to learn both local features and global topological properties by iteratively propagating node information across the edges of the graph.

Graph Convolutional Networks. A popular instantiation of the GNN framework is the Graph Convolutional Network (GCN) (Kipf et al., 2017), which generalize the concept of convolutions from regular grids to graphs. To ensure numerical stability, GCNs typically assume no node self-loops and modify the adjacency matrix as $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. It is then transformed into the symmetric normalized adjacency matrix $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ to prevent the scale of feature vectors from exploding or vanishing during deep stacking. Here, $\tilde{\mathbf{D}}$ is the diagonal degree matrix of $\tilde{\mathbf{A}}$ with entries $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. GCNs utilize this normalized operator to define a layer-wise update rule. Utilizing a non-linear activation function σ and a trainable weight matrix $\mathbf{W}^{(l)}$ at layer l, the update is defined as

$$\mathbf{X}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}^{(l)}\mathbf{W}^{(l)}\right).$$

This formulation is permutation equivariant by design, and the normalization effectively performs a weighted local averaging of features. Reformulating this update rule in terms of the neighborhood of each node and using node features $\mathbf{h}_\mathcal{V} : \mathcal{V} \to \mathbb{R}^{d_\mathcal{V}}$ of dimension $d_\mathcal{V}$, we can express the vector-valued update for a single node v as

$$\mathbf{h}_v^{l+1} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} c_{uv}\mathbf{h}_u^l\mathbf{W}^{(l)}\right) \tag{2.12}$$

with $c_{uv} = (\tilde{D}_{uu}\tilde{D}_{vv})^{-1/2}$ as the normalization constant.

Intuitively, the GCN update rule acts as a spatial low-pass filter, effectively smoothing features across the graph topology. This property is desirable for homophilic tasks where neighboring nodes are expected to have similar features. However, the fixed nature of the adjacency matrix in GCNs can limit their expressiveness, as they cannot learn to weigh the importance of different neighbors differently.

Graph Attention Networks. Graph Attention Networks (GATs) (Velickovic et al., 2018) introduce an attention mechanism to the node aggregations, allowing each node to individually weigh the importance of its neighbors. Mathematically, the update rule for a GAT layer for a node v is given by

$$\mathbf{h}_v^{l+1} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \alpha_{uv} \mathbf{h}_u^l \mathbf{W}^{(l)}\right), \tag{2.13}$$

where α_{uv} are normalized attention coefficients that determine the importance of node u's features to node v's update. This update rule recovers Equation 2.12 when setting $\alpha_{uv} \propto c_{uv}$ up to normalization and is thus generally more expressive than GCNs. However, it still depends on a scalar attention coefficient for each neighbor, which can limit its ability to capture complex interactions between nodes. Additionally, the standard GAT formulation does not consider edge features, which are crucial for applications involving geometric graphs, i.e., graphs in some Euclidean space, such as those used for the mesh-based simulations discussed in this work.

Message Passing Networks. Message Passing Networks (MPNs) (Pfaff et al., 2021; Linkerhägner et al., 2023) further generalize the notion of local information aggregation on nodes. Compared to GCNs, which use a fixed normalized adjacency matrix to aggregate neighboring features, and GATs, which learn scalar attention coefficients for each neighbor, MPNs allow for arbitrary learned messages along edges. By directly operating on edges, they also trivially incorporate edge features $\mathbf{h}_{\mathcal{E}} : \mathcal{E} \to \mathbb{R}^{d_{\mathcal{E}}}$ of dimension $d_{\mathcal{E}}$ into the message-passing process, which is essential for learning on geometric graphs. As such, they are a popular architecture for mesh-based physical simulation (Pfaff et al., 2021; Linkerhägner et al., 2023; Würth et al., 2025; Dahlinger et al., 2025).

Intuitively, MPNs operate by exchanging information along the edges of a graph to iteratively update node representations. This process allows the network to capture complex dependencies within the graph structure, mimicking the function class of several classical PDE solvers (Brandstetter et al., 2022). Using this message passing structure, the model iteratively updates high-dimensional latent representations over L discrete steps. Starting from learned linear embeddings $\mathbf{h}_v^0 = \mathbf{h}_v\mathbf{M}_v$ and $\mathbf{h}_e^0 = \mathbf{h}_e\mathbf{M}_e$ of the initial features, each message passing step l computes:

$$\mathbf{h}_e^{l+1} = \mathbf{h}_e^l + \psi_{\mathcal{E}}^l(\mathbf{h}_v^l, \mathbf{h}_u^l, \mathbf{h}_e^l), \text{ with } e = (u, v), \qquad \mathbf{h}_v^{l+1} = \mathbf{h}_v^l + \psi_{\mathcal{V}}^l(\mathbf{h}_v^l, \bigoplus_{e=(v,u) \in \mathcal{E}} \mathbf{h}_e^{l+1}). \tag{2.14}$$

The permutation-invariant aggregation $\bigoplus$ is typically realized via sum, mean, or maximum operators. The functions $\psi^l_\mathcal{E}$ and $\psi^l_\mathcal{V}$ are learnable components, usually parameterized as Multilayer Perceptrons (MLPs). The inclusion of the current states $\mathbf{h}^l_v$ and $\mathbf{h}^l_e$ in the update rules acts as a residual connection and accounts for self-interaction, typically removing the need for explicit self-loops. To recover Equation 2.13, we assume the existence of self-loops $(v, v) \in \mathcal{E}$ and set $\bigoplus = \sum$. We then let $\psi^l_\mathcal{E}(\mathbf{h}^l_v, \mathbf{h}^l_u, \mathbf{h}^l_e) = \alpha_{uv}\mathbf{h}^l_u\mathbf{W}^{(l)} - \mathbf{h}^l_e$ to negate the edge residual, and use $\psi^l_\mathcal{V}$ to apply the non-linear activation σ to the aggregate and cancel out the node-level residual $\mathbf{h}^l_v$.

The output of the final MPN layer is a learned representation $\mathbf{h}^L_v$ for each node $v \in \mathcal{V}$. In the context of this thesis, we feed this representation into a decoder MLP to yield one or multiple predictions $x_j = \text{MPN}(\mathcal{G}, \mathbf{h}_\mathcal{V}, \mathbf{h}_\mathcal{E})_j$ per node $v_j \in \mathcal{V}$, which we denote as MPN(v_j) for brevity. These predictions can be used for various downstream algorithms. For example, we utilize MPN(v_j) to parameterize policy actions for mesh refinement in Chapter 4 and to predict sizing fields for mesh generation in Chapter 5. In both cases, the adaptive meshing algorithms facilitate efficient simulation of physical systems by providing high-quality discretizations to downstream numerical solvers.

3. Inferring Versatile Behavior from Demonstrations by Matching Geometric Descriptors

Learning from human demonstrations is a powerful paradigm for teaching robots complex behaviors without the need for explicit reward engineering. However, human behavior is often inherently versatile, and capturing this multi-modality remains a significant challenge. Standard Imitation Learning (IL) approaches, as introduced in Section 2.1.1 often struggle with such data, as they either average over distinct modes of behavior or fail to generalize learned strategies to novel task configurations. This difficulty is often rooted in the choice of representation, as absolute coordinates and raw state-action traces vary significantly even when the underlying geometry of the task remains similar. This chapter proposes Versatile Imitation from Geometrically Observed Representations (*VIGOR*), a novel discriminator-based IL method, to alleviate these issues and answer the first research question of this thesis, i.e.,

Q1 How can we utilize geometric representations to capture versatile human expert behavior in robotic manipulation?

At its core, *VIGOR* encodes trajectories as geometric relations over time, such as distances and orientations relative to task-relevant objects (Englert et al., 2017; Englert et al., 2018). Such a geometric space abstracts away from concrete robot configurations, transforming unstructured motion data into a representation where the essential task structure becomes invariant to the specific context. This transformation allows *VIGOR* to produce versatile policies that match the expert in a high-level descriptor space and generalize across different contexts and task variations.

To handle the multi-modality inherent in human demonstrations, even in descriptor space, *VIGOR* utilizes a decomposition similar to Variational Inference by Policy Search (VIPS) (Arenz et al., 2018; Arenz et al., 2020) to optimize the Information-Projection, which we introduce in Section 2.1.3. This objective minimizes the relative entropy from the learner's policy to the expert's distribution. Concretely, *VIGOR* is based on Expected Information Maximization (EIM) (Becker et al., 2020), which is a sample-based extension of this decomposition. Unlike commonly used mode-averaging objectives like maximum likelihood, this mode-seeking objective facilitates capturing distinct, non-overlapping expert behaviors. We employ this framework to train expressive trajectory-level Gaussian Mixture policies that can represent complex, multi-modal solution strategies.

The following work was published as **Inferring Versatile Behavior from Demonstrations by Matching Geometric Descriptors** *(Niklas Freymuth, Nicolas Schreiber, Philipp Becker, Aleksandar Taranovic, Gerhard Neumann) in the 6th Conference on Robot Learning, CoRL 2022. Reprinted with permission of the authors. Wording, notation, structure and formulations were revised in several places.*

3.1. Introduction

IL (Schaal, 1996; Argall et al., 2009; Hussein et al., 2017) from human demonstrations is challenging as humans often solve tasks in versatile ways. Even the same person might solve a task differently when confronted with it multiple times. Behavior can be versatile in terms of task-level decisions, such as planning a route to a target, and individual actions, such as randomly pausing during a movement to think about the next steps. Most recent IL approaches (Fu et al., 2018; Ho et al., 2016; Brown et al., 2019; Brown et al., 2020) model the behavior in state-action space using Gaussian policies, which assume the behavior is unimodal and cannot capture this planning versatility. Yet, this assumption is violated by most human expert datasets, often causing poor generalization to human demonstrations (Orsini et al., 2021). Additionally, IL approaches often need immense amounts of data to achieve generalization (Osa et al., 2018; Orsini et al., 2021). We tackle these challenges with a feature-matching approach to IL that is able to generalize to novel contexts from a small number of expert demonstrations. Such contexts are (typically low-dimensional) vectors that describe a specific task configuration within a family of related tasks, such as the coordinates of goal locations to reach. The resulting approach, Versatile Imitation from Geometrically Observed Representations (*VIGOR*), models distributions that match expert trajectories in terms of concise geometric behavioral descriptors. *VIGOR* represents distributions using Gaussian Mixture Models (GMMs) over Probabilistic Motion Primitives (ProMPs) (Paraschos et al., 2013), and utilizes descriptors that are designed to abstract away from concrete contexts and thus facilitate generalization to novel contexts in a sample-efficient way. An example of such descriptors would be geometric features in the form of (only) the distance between a target and the robot's end-effector, plus features that denote the smoothness of the robot's motion. While similar behavioral descriptors have found use in Inverse Optimal Control (Englert et al., 2017; Englert et al., 2018) to recover cost functions from demonstrations, common IL approaches instead use state representations that include both relative and absolute information (Ho et al., 2016; Mandlekar et al., 2021). The exact form of this set of behavioral descriptors is task-dependent and allows the user to include domain knowledge. Figure 3.1 shows how *VIGOR* works on an exemplary reaching task. For training the GMMs, we rely on recent approaches (Arenz et al., 2018; Arenz et al., 2020; Becker et al., 2020) for distribution matching and Variational Inference. For inference, we train individual GMMs for each training context and an arbitrary number of test contexts.

We evaluate *VIGOR* on a suite of versatile robotic tasks, where demonstrations are collected from human experts through teleoperation. We compare our method to standard

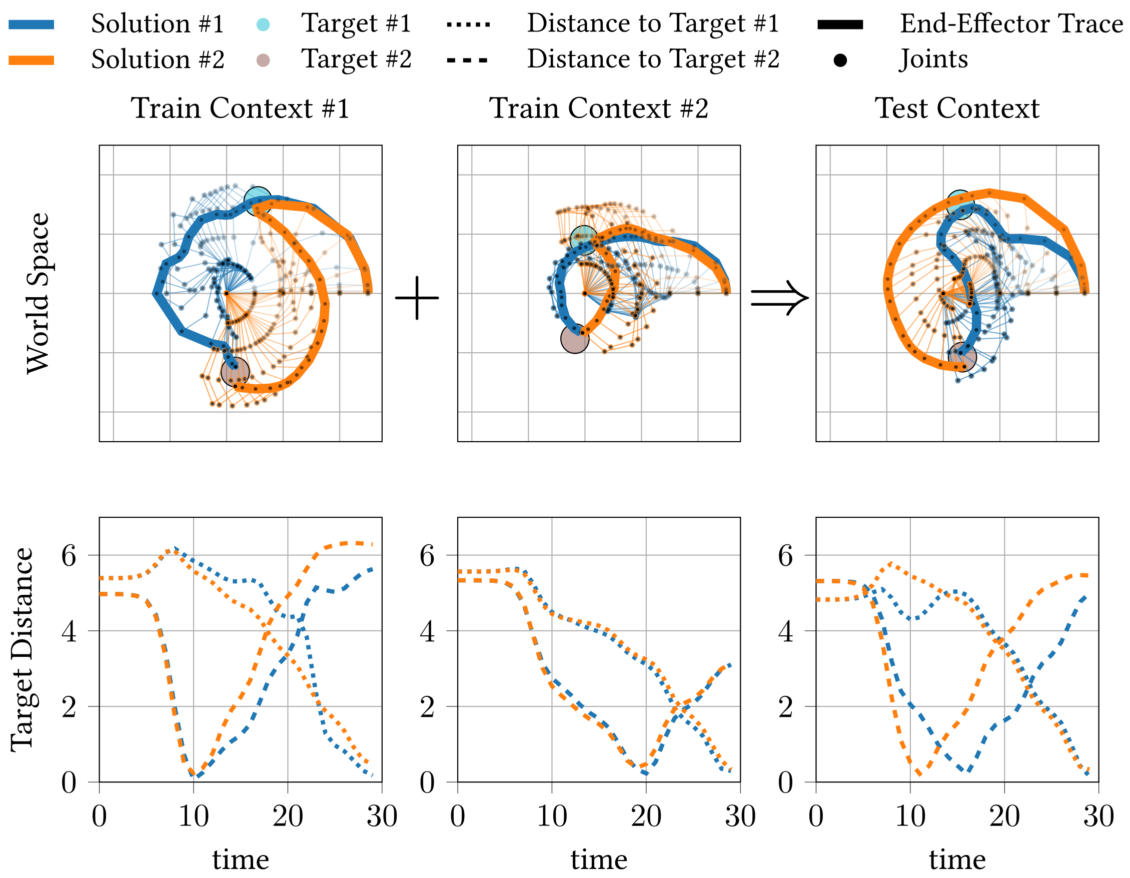


Figure 3.1.: End-effector traces (**top**) and distances to target centers (**bottom**) for a planar point-reaching task on 3 different contexts. (**Left**, **Middle**:) Human experts solve the task by reaching the blue target and ending their trajectory in the grey one. (**Right**:) *VIGOR* matches distributions of behavioral descriptors, such as the depicted target distances, of these demonstrations. This produces versatile behavior in unseen contexts, such as new target positions. Depicted are 2 different component means from a trained GMM policy. For each context, we show the joint configurations for the first trajectory in blue and the second trajectory in orange.

Behavioral Cloning (BC) (Bain et al., 1995), BC with a GMM policy (Mandlekar et al., 2021; Zhou et al., 2020), Generative Adversarial Imitation Learning (GAIL) (Ho et al., 2016) and Inverse Reinforcement Learning (IRL) techniques (Brown et al., 2020). We find that these methods either fail to learn useful behavior because they average over multiple solutions, or that they are unable to capture the full versatility of the demonstrations. Contrary to this, *VIGOR* accurately models highly versatile behavior from an already small number of trajectories demonstrated by human experts. We perform extensive parameter studies to showcase the importance of different parameters and design choices.

To summarize, our contributions are:

Datasets and code can be found at `https://www.github.com/NiklasFreymuth/VIGOR`.

- We infer *multi-modal distributions* of desired trajectories from highly versatile human expert trajectories by matching distributions over behavioral descriptors between learner and expert.
- Our approach is, to the best of our knowledge, the first Adversarial Imitation Learning method to utilize *concise behavioral descriptors* to facilitate generalization to novel contexts from as few as a small number of demonstrations.
- We conduct *experiments with human demonstrations* in simulation and on a real robot and find that *VIGOR* accurately models highly versatile human behavior, outperforming various Imitation Learning baselines.

3.2. Related Work

3.2.1. Imitation Learning for Skills

A common way to represent skills over trajectories is via Movement Primitives (MPs) (Schaal, 2006; Paraschos et al., 2013). While learning individual MPs from human demonstrations is a simple regression problem, modeling versatile behavior requires a more sophisticated model that can represent such multi-modality, e.g., a mixture model. Using Expectation Maximization (EM) (Dempster et al., 1977), both (Mülling et al., 2013) and (Ewerton et al., 2015) fit a single GMM over MP parameters of multiple demonstrations for different contexts and use them to generalize to novel contexts by conditioning. Yet, as they only fit a single GMM for all contexts, they do not explicitly focus on representing versatility. Recent work (Pervez et al., 2018) instead learns a mixture over contextualized MPs using Gaussian Mixture Regression. This has been extended to non-linear relations between MP parameters and the context (Zhou et al., 2020) by using Mixture Density Networks (MDNs) (Bishop, 1994). While the above approaches work for tasks with a small number of modes, optimizing MDNs can be challenging in the case of many modes in the demonstrations. Osa et al. (2017) use planning algorithms to reproduce human demonstrations in a trajectory-based setting. They tune the parameters of a cost function such that the planned trajectories match the demonstrations and use the extracted cost function to guide the trajectory optimization process. Yet, this approach is limited to uni-modal demonstrations and requires engineered cost functions.

3.2.2. Imitation Learning by Distribution Matching

Classical IL approaches (Abbeel et al., 2004; Ziebart et al., 2008) employed feature expectation matching to learn a policy from behavioral descriptors of expert demonstrations. Similarly, a recent body of work (Englert et al., 2017; Englert et al., 2018) in Inverse Optimal Control utilizes geometric behavioral descriptors that are similar to ours to learn cost functions for manipulation tasks from a few demonstrations. While these approaches linearly match the moments/expectations of their features, our method instead matches a versatile distribution over non-linear features using the reverse Kullback-Leibler (KL) divergence (Kullback et al., 1951). Directly matching the distribution rather than its moments means that our method can represent complex and multi-modal distributions. More recently, a new class of distribution matching approaches has gained popularity with the advent of Generative Adversarial Nets (Goodfellow et al., 2014). Starting from GAIL (Ho et al., 2016) a whole class of distribution matching based IL approaches was developed (Fu et al., 2018; Kostrikov et al., 2018; Torabi et al., 2019; Kostrikov et al., 2019; Ghasemipour et al., 2020; Zhang et al., 2020a). All of these approaches commonly work in a step-based setting and minimize some divergence with respect to the state-action occupancy or state marginals. They generally do not consider versatile behavior, and additionally need to interact with the environment during training in order to generate states to match. We instead work in a trajectory-based setting, which allows us to abstract away the environment's dynamics and learn entirely offline, i.e., without any environment interactions. Common methods in this setting usually employ a maximum likelihood approach, which, as previously discussed, performs poorly on versatile data. Recent work (Becker et al., 2020) proposes using the Information-Projection (Murphy, 2012) instead, which is able to focus on individual modes of data rather than averaging over all modes of data. This approach has been extended to IRL (Freymuth et al., 2021). Here, we build on these methods, generalizing them to sequential data and geometric behavioral descriptors.

3.2.3. Comparison-Based Approaches

Brown et al. (2019) perform IRL by utilizing provided rankings of demonstrations to train a discriminator on a comparison-based loss. As the discriminator learns to favor samples with a high rank, it can be used as a reward function after training. This has been extended by Disturbance-based Reward Extrapolation (D-REX) (Brown et al., 2020), which automatically generates rankings by fitting a BC policy on the demonstrations and subsequently draws samples with different noise levels from this policy. We adapt D-REX to our trajectory-based setting as a baseline. Similar to *VIGOR*, this adapted method trains a model on expert demonstrations on training contexts to optimize GMM policies on novel test contexts. However, while D-REX learns a reward function to do so, *VIGOR* iteratively (re-)trains a discriminator on the expert demonstrations and policy samples.

3.3. Inferring Versatile Behaviors by Matching Geometric Descriptors

Our approach estimates versatile behavior in the form of GMM distributions for sets of training and test contexts of a given task. We note that while this setup requires the test contexts in advance, it does not need interaction with the real environment since the behavioral descriptors are directly computed from the proposed planned trajectory and its context. To capture correlations in the demonstrations, we utilize full-covariance GMM components. For each planned trajectory, we compute concise descriptors such as

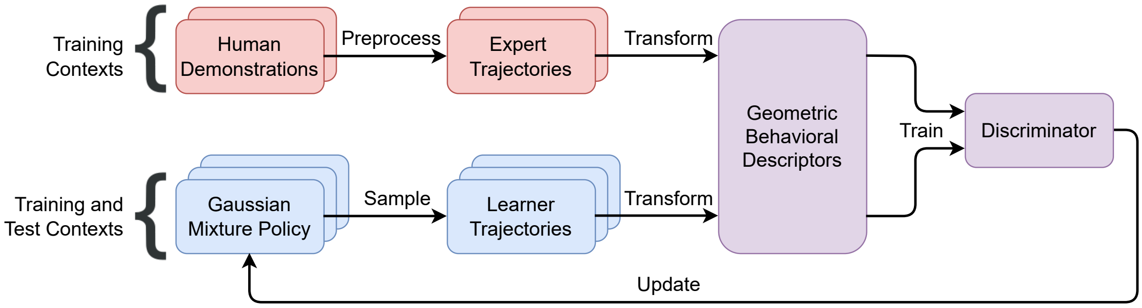


Figure 3.2.: *VIGOR* generates a set of expert trajectories from human demonstrations and transform them into geometric behavioral descriptors. These descriptors are then fed into a discriminator. This process is repeated for samples of a separate GMM policy for each training and test context. We then train the discriminator to distinguish between human and policy trajectories. The geometric descriptors are designed to abstract away from concrete task configurations, causing the discriminator to distinguish how well a trajectory performs rather than which context it acts on. As a result, the discriminator can be used to improve the policies.

distances to key-points that isolate the performance of the trajectory from its context to allow for sample-efficient generalization to novel contexts. We then build on Expected Information Maximization (EIM) (Becker et al., 2020) and use a discriminator to infer GMMs from demonstrations by matching the learner's and expert's distributions in the resulting geometric feature space. Towards this end, we generalize EIM to sequential data and modify the policy parameterization to accommodate for our setting. Figure 3.2 illustrates an overview of our approach.

3.3.1. Probabilistic Movement Primitives

To facilitate versatility over the full task duration, we represent trajectories $\boldsymbol{\tau} = (\boldsymbol{\tau}_1, \cdots, \boldsymbol{\tau}_T)$ using Probabilistic Motion Primitives (ProMPs). Here, each step is represented as $\boldsymbol{\tau}_t = \boldsymbol{\Phi}(t)^T\mathbf{w}$, where $\mathbf{w}$ denotes the parameters of the ProMP and $\boldsymbol{\Phi}(t)$ are time-dependent features, usually radial basis functions centered around different time points (Zhou et al., 2020). For a single demonstration, the parameters $\mathbf{w}$ are fitted using simple linear regression and compactly represent the entire trajectory. Note that $\mathbf{w}$ does not depend on t and that $\boldsymbol{\tau}$ can thus be computed at an arbitrary resolution T.

Variational Inference for Gaussian Mixture Models. We build on a recent class of efficient Variational Inference approaches that allow modeling versatile behavior with GMMs (Becker et al., 2020; Arenz et al., 2018). These approaches use the Information-Projection (Murphy, 2012) to minimize the reverse Kullback-Leibler (KL) divergence $\mathrm{KL}\left(q(\mathbf{w}) \parallel p(\mathbf{w})\right)$ between a model $q(\mathbf{w})$ and a target distribution $p(\mathbf{w})$. In our case, $p(\mathbf{w})$ is the distribution of expert trajectories. For this setting, Variational Inference by Policy Search (VIPS) (Arenz et al., 2018; Arenz et al., 2020) introduces an upper-bound objective based on a variational decomposition, as detailed in Section 2.1.3. In our setting, VIPS repeatedly minimizes

$$q\left(\mathbf{w}\right) = \underset{q(\mathbf{w})}{\arg\min}\, \mathbb{E}_{q(\mathbf{w},z)}\left[\log \frac{\hat{q}\left(\mathbf{w}\right)}{p\left(\mathbf{w}\right)}\right] + \mathrm{KL}\left(q(z) \parallel \hat{q}(z)\right) + \mathbb{E}_{q(z)}\Big[\mathrm{KL}\left(q\left(\mathbf{w}|z\right) \parallel \hat{q}\left(\mathbf{w}|z\right)\right)\Big], \tag{3.1}$$

where $\hat{q}$ denotes the model from the previous iteration. This objective decomposes into individual optimizations for the components and categorical distributions. The latter are solved using trust-region methods from policy search (Peters et al., 2010; Abdolmaleki et al., 2015).

3.3.2. Distribution Matching for Gaussian Mixture Models

In order to infer a GMM policy $q(\mathbf{w})$ that matches the expert's behavior, we can utilize Expected Information Maximization (EIM) (Becker et al., 2020). EIM proposes using density ratio estimation (Sugiyama et al., 2012) approaches, specifically logistic regression, to approximate the $\log(\hat{q}(\mathbf{w})/p(\mathbf{w}))$ term in Equation 3.1. This change removes the dependency on $p(\mathbf{w})$, and resembles Generative Adversarial Nets (Goodfellow et al., 2014) where the discriminator effectively also estimates a density ratio.

3.3.3. Context-Specific Mixture Policies

We want to model the expert's behavior for a given small set of training configurations of the task, denoted as training contexts $\mathbf{c}_{\text{train}}$, and a set of unseen test contexts $\mathbf{c}_{\text{test}}$. We consider the set of all contexts as $\mathbf{c} = \mathbf{c}_{\text{train}} \cup \mathbf{c}_{\text{test}}$. As context-dependent GMMs are hard to learn and we assume a relatively small amount of training contexts, 6 in all our experiments, with multiple demonstrations per training context, we resort to maintaining a single GMM-Policy $q_c(\mathbf{w})$ for each context c. This non-amortized formulation simplifies the policy updates as we can optimize the policy for each context individually, given the density ratio estimator, using highly efficient, tailored methods for full-rank GMM approximations (Arenz et al., 2018; Arenz et al., 2020).

Such GMMs could in principle be represented by MDNs. However, attempts with MDN and full-covariance Gaussian components were highly unsuccessful in our experiments.

3.3.4. Distribution Matching of Behavioral Descriptors

We assume access to behavioral descriptors $\mathbf{O}_w = f_c(\mathbf{w})$ that represent features encoded in the parameter vector $\mathbf{w}$ with respect to the context c of $\mathbf{w}$. We want to match the distribution of behavioral descriptors of the demonstrator while optimizing for $q(\mathbf{w})$, i.e,

$$q^*(\mathbf{w}) = \underset{q(\mathbf{w})}{\arg\min}\, \mathrm{KL}(q\left(\mathbf{O}_w\right)||p(\mathbf{O}_w)) = \underset{q(\mathbf{w})}{\arg\min}\, \mathbf{E}_{q(\mathbf{w})}\left[\int_{\mathbf{O}_w} p(\mathbf{O}_w|\mathbf{w}) \log \frac{q(\mathbf{O}_w)}{p(\mathbf{O}_w)} d\mathbf{O}_w\right],$$

with $q(\mathbf{O}_w) = \int p(\mathbf{O}_w|\mathbf{w})q(\mathbf{w})d\mathbf{w}$. For simplicity, we assume a deterministic mapping between $\mathbf{w}$ and $\mathbf{O}_w$, which reduces $p(\mathbf{O}_w|\mathbf{w})$ to a Dirac delta. With this deterministic mapping, we can directly apply EIM to the above objective, the only difference being that the discriminator is trained using $\mathbf{O}_w$ instead of $\mathbf{w}$. Moreover, while we learn different distributions $q_c(\mathbf{w})$ for each context c, we use the same discriminator for all contexts. This way, we can infer distributions $\{q_c(\mathbf{w})|c \in \mathbf{c}_{\text{test}}\}$, i.e. infer versatile trajectories for unseen scenarios. In the following, we use $\mathbf{O}$ as a shorthand for $\mathbf{O}_w$ for brevity. Algorithm 1 provides a high-level overview of the method.

3.3.5. Density Ratio Estimation for Sequential Behavioral Descriptors

As in our case, $\mathbf{w}$ encodes the desired trajectory $\boldsymbol{\tau}$, the behavioral descriptors are typically computed per time-step, i.e., $\mathbf{O} = (\mathbf{o}_1, \dots, \mathbf{o}_T)^T$, where each $\mathbf{o}_t$ is a feature vector. To enable the classifier to deal with sequential data, we consider a sequence-to-sequence neural network $\boldsymbol{\psi}(\mathbf{O}) = \boldsymbol{\psi}((\mathbf{o}_1, \dots, \mathbf{o}_T)^T) = (y_1, \dots, y_T)^T$. The network receives a sequence of inputs $\mathbf{O} = (\mathbf{o}_1, \dots, \mathbf{o}_T)^T$ and outputs a sequence of values $(y_1, \dots, y_T)^T$, where each y_i may depend on multiple $\mathbf{o}_j$. In each iteration of our optimization, we (re-)train this network to discriminate between the provided demonstrations $\mathbf{O}^{(p)}$ of $\mathbf{c}_{\text{train}}$, and an equal number of policy samples $\mathbf{O}^{(q)}$ drawn uniformly from $\mathbf{c}$. We first consider the case where we want to discriminate full trajectories as in EIM. Denoting the sigmoid function as σ, the discriminator can straightforwardly be trained on a binary cross-entropy loss of the sum of sequence values $\hat{y} = \sum_{t=0}^{T} y_t$, i.e., by minimizing

$$\mathrm{BCE}(\boldsymbol{\psi}(\mathbf{O}), \mathbf{O}^{(p)}, \mathbf{O}^{(q)}) = -\mathbf{E}_{\mathbf{O}^{(q)}}\left[\log\left(\sigma\left(\hat{y}\right)\right)\right] - \mathbf{E}_{\mathbf{O}^{(p)}}\left[\log\left(1 - \sigma\left(\hat{y}\right)\right)\right], \tag{3.2}$$

with respect to $\boldsymbol{\psi}(\mathbf{O})$. Equation 3.2 recovers the log density ratio $\boldsymbol{\psi}(\mathbf{O}) = \log \mathbf{O}^{(p)} - \log \mathbf{O}^{(q)}$ at convergence (Sugiyama et al., 2012). However, this cost function only provides a single training sample per trajectory and is therefore hard to use for a small number of demonstrations. By utilizing the sequential structure of the trajectories, the classification error can be computed for each time-step, i.e. we minimize

$$\mathrm{BCE}_{\text{step}}(\boldsymbol{\psi}(\mathbf{O}), \mathbf{O}^{(p)}, \mathbf{O}^{(q)}) = -\mathbf{E}_{\mathbf{O}^{(q)}}\left[\sum_{t=0}^{T} \log\left(\sigma\left(y_t\right)\right)\right] - \mathbf{E}_{\mathbf{O}^{(p)}}\left[\sum_{t=0}^{T} \log\left(1 - \sigma\left(y_t\right)\right)\right] \tag{3.3}$$

Algorithm 1: *VIGOR*

Input: Contexts $\mathbf{c} = \mathbf{c}_{\text{train}} \cup \mathbf{c}_{\text{test}}$
Input: Mappings f_c with $\mathbf{O} = f_c(\mathbf{w})$ for each $c \in \mathbf{c}$
Input: Expert Descriptors $\mathbf{O}^{(p)} = \{\mathbf{O}_c^{(p)} | c \in \mathbf{c}_{\text{train}}\}$
Input: Initial Policies $\{q_c(\mathbf{w}) | c \in \mathbf{c}\}$
Output: Converged Policies $\{q_c(\mathbf{w}) | c \in \mathbf{c}\}$

$n_{\text{samples}} \leftarrow |\mathbf{O}^{(p)}|/|\mathbf{c}|$
$\hat{q}_c(\mathbf{w}) \leftarrow q_c(\mathbf{w}) \quad \forall c \in \mathbf{c}$
while *not converged* **do**
  Gather new policy samples
  $\mathbf{O}^{(q)} \leftarrow \{\}$
  for $c \in \mathbf{c}$ **do**
    $\hat{\mathbf{O}}_c^{(q)} \leftarrow \{\mathbf{w}_c^{(j)}\}_{j=1\dots n_{\text{samples}}} \sim q_c(\mathbf{w})$
    $\mathbf{O}^{(q)} \leftarrow \mathbf{O}^{(q)} \cup \{f_c(\mathbf{w}) | \mathbf{w} \in \hat{\mathbf{O}}_c^{(q)}\}$
  end
  Train discriminator $\psi(\mathbf{O})$ (Equation 3.3)
  $\psi(\mathbf{O}) \leftarrow \arg\min_{\psi(\mathbf{O})} \text{BCE}(\psi(\mathbf{O}), \mathbf{O}^{(p)}, \mathbf{O}^{(q)})$
  Update policies per context with discriminator $\psi(\mathbf{O}) \approx \log \mathbf{O}^{(p)} - \log \mathbf{O}^{(q)}$ (Equation 3.1)
  for $c \in \mathbf{c}$ **do**
    $q_c(\mathbf{w}) \leftarrow \arg\min_{q(\mathbf{w})} \mathbb{E}_{q(\mathbf{w},z)}\left[\sum_t \psi(\mathbf{O})\right] + \text{KL}(q_c(z) \parallel \hat{q}_c(z)) + \mathbb{E}_{q(z)}\left[\text{KL}(q(\mathbf{w}|z) \parallel \hat{q}_c(\mathbf{w}|z))\right]$
    $\hat{q}_c(\mathbf{w}|z) \leftarrow q(\mathbf{w}|z) \quad \forall z$
    $\hat{q}_c(\mathbf{w}) \leftarrow q_c(\mathbf{w})$
  end
end
return $\{q_c(\mathbf{w}) | c \in \mathbf{c}\}$

with respect to $\psi(\mathbf{O})$. Finally, we train an ensemble of discriminators instead of a single one, using their average logit as the log density ratio estimate for the policy updates. For the network architecture, we find that simple $2-4$ layer $1d$ Convolutional Neural Networks (CNNs) work best in our experiments.

3.3.6. Geometric Behavioral Descriptors

VIGOR utilizes geometric descriptors that abstract away from a concrete task configuration to generalize to new configurations (Englert et al., 2017; Englert et al., 2018). To this end, we encode the current state along the desired trajectory with respect to relative geometric features rather than absolute values. The resulting geometric descriptors cause trajectories of similar performance to appear similar in feature space regardless of their context, allowing for sample-efficient generalization to novel contexts. Such a descriptor space can often be straightforwardly constructed from the geometry of the task by composing distances of the end-effector to key-points of the objects in a scene. For example, in object manipulation, this space can be composed of distances of the end-effector to the corners of a box to push.

3.4. Experiments

All experiments use human demonstrations. The tasks were selected due to their highly multi-modal nature and because they allow for an easy collection of human demonstrations. We instructed the demonstrators to solve the task in varying ways to create versatile solutions. As a preprocessing step, we fit a ProMP for each human demonstration and filter the resulting ProMPs such that only successful demonstrations remain. This step ensures that the human demonstrations can be imitated by the learner and conveniently allows us to compress all demonstrations to a fixed length in a principled fashion. We use the resulting preprocessed expert demonstrations for all experiments unless otherwise stated. Similarly, unless noted otherwise, all experiments use 6 training and 6 test contexts, and 5 GMM components for each configuration and method, yielding a total of 30 expert demonstrations for training. Experimentally, this number of components is sufficient for matching the distribution of expert descriptors for the considered tasks, whereas more components generally only lead to spurious improvements. We experiment with fewer components in the parameter studies in Section A.1. Unless noted otherwise, we do *not* train the categorical distribution of the GMM components, using a uniform distribution instead. We evaluate the GMM policies using a number of samples for each component of each *test* context, and then rank these components according to the average performance of their samples. We then report statistics of the best component of the resulting value over the random seeds. Note that reporting the best component leads to a fair comparison to unimodal approaches. For baselines that do not use GMM policies, we instead draw samples of the trained policies for each test context and average the performance of all of them. We repeat each experiment for 10 random seeds. Section A.2 provides hyperparameters and training details.

3.4.1. Baselines

We compare *VIGOR* to a number of strong IL baselines in trajectory-based and state-action settings.

EM+D-REX. We modify Disturbance-based Reward Extrapolation (D-REX) (Brown et al., 2020), a recent ranking-based IL algorithm, to work in a trajectory-based setting and with a GMM policy. For this, we first use EM to fit a GMM with a small number of components on each configuration in $\mathbf{c}_{\text{train}}$, and use the resulting distributions to draw samples for the D-REX reward. To generate the rankings required by D-REX, we multiply the covariance of each component of the resulting GMM with different scalars, resulting in GMMs with different noise levels. More precisely, we fit the EM-GMM on the samples, and multiply its component-wise covariance with a *Base Noise* σ_{base} to get an initial distribution. We repeat this process for a number of *Noise Levels*, linearly increasing the noise up to $\sigma_{\text{base}} \cdot \lambda_{\text{max}}$, where we call λ_{max} the *Noise Multiplier*. We then draw samples from each noise level of the GMMs corresponding to each context in $\mathbf{c}_{\text{train}}$ and rank them against each other using the comparison-based loss of D-REX, where samples drawn from a lower noise level are ranked higher. The resulting ranked demonstrations are transformed into geometric descriptors $\mathbf{O}$ and used to train a reward function $R(\mathbf{O})$, which in turn is used by a policy optimization algorithm to optimize policies on unknown contexts $\mathbf{c}_{\text{test}}$. For the policy optimization algorithm, we use VIPS (Arenz et al., 2018; Arenz et al., 2020) to ensure a fair comparison with *VIGOR*. We further introduce a *Reward Scale*, optimizing $\widehat{R}(\mathbf{O}) = \alpha_{\text{scale}} R(\mathbf{O})$ with a scalar α_{scale} to stabilize training. We call the resulting algorithm *EM+D-REX*. Comparing *EM+D-REX* and *VIGOR*, both approaches train a discriminator that uses expert demonstrations on training contexts to fit novel test contexts. Similarly, both make use of VIPS to optimize marginal policies to match multi-modal distributions over geometric behavioral descriptors. However, *EM+D-REX* uses the discriminator to represent a reward function, while *VIGOR* iteratively re-trains the discriminator until convergence of the test policies.

State-Action Baselines. Next, we compare to common state-action based imitation learning algorithms to see how well these approaches work on versatile human demonstrations. We experiment with BC (Bain et al., 1995) and *GAIL* (Ho et al., 2016) as implemented in the `Imitation` (Wang et al., 2020a) repository. These state-action baselines use velocities as actions, and receive the geometric descriptors of the current robot state plus the current time-step as state information. For the Point Reaching tasks, we also experiment with encoding which of the points has been reached to make the tasks Markovian. To accommodate for versatility on an action level, we also compare to a variant of BC whose policy head outputs GMM parameters over actions per state (Mandlekar et al., 2021). We do not make use of the Low Noise Evaluation Trick proposed in Mandlekar et al. (2021), which leads to minor improvements in their experiments and does not affect our results. During evaluation, we also generate each trajectory from a fixed component rather than sampling a new component mean per step for consistency with the other approaches. The resulting methods are called *BC(S)* and *BC-GMM(S)* respectively, where the suffix '(S)' indicates that these are state-action based methods. To prevent covariance collapse, both *BC* and *BC-GMM* also make use of an auxiliary entropy regularization term (Zhou et al., 2020). We use the Proximal Policy Optimization (PPO) (Schulman et al., 2017) implementation of `Stable Baselines3` (Raffin et al., 2021) as the *GAIL* policy. All state-action baselines use regular Multilayer Perceptrons (MLPs) for their policy.

Trajectory-based Baselines. We further propose trajectory-based baselines that imitate full trajectories instead of individual state-action pairs. Using a suffix '(T)' to denote trajectory-level baselines, these are *BC(T)* and *BC-GMM(T)*. Both baselines are added to see how simple (multi-modal) behavioral cloning works on full expert trajectories. In this setup, the methods see as input the context of the task, e.g., the position of the target points for the point reaching tasks, and output a contextual distribution $q(\mathbf{w}|c)$ over ProMP parameters. This distribution is implemented as a simple MLP.

3.4.2. `Planar Reacher`

First, we consider the introductory task shown in Figure 3.1. The goal is to reach an intermediate target area with the end-effector of a 5 Degrees of Freedom (DoF) robot with joints of length 1 before ending the trajectory in a final target area. The robot always starts with all joints at a resting position of $0°$. Each context is specified by its target positions, which are drawn from independent isotropic Gaussians with means $(0.5, 2.5)$ and $(0.5, -2.5)$ and standard deviation 0.5. Both targets have a radius of $r{=}0.5$ to be considered *reached*. For evaluation, we use the *Border Distance* of the target areas, using the minimum distance over time for the first target and the distance at the last time-step for the second target. We further call a demonstration successful if its *Border Distance* is 0. We collect demonstrations via a joystick-based setup to control the end-effector's (x, y)-position and rotation, and reconstruct joint angles using an inverse kinematics controller. We train all methods on 5 demonstrations per training context.

The geometric descriptors for this task are given by the Euclidean distance of the end-effector to both targets, paired with the average velocity and acceleration of the joints to encode the smoothness of the motion. This process yields four features per time-step. The left and middle panels of Figure 3.1 depict example demonstrations for two training contexts (**top**) and the corresponding distances between the end-effector and targets over time (**bottom**). Section A.1 explores other choices for the geometric descriptors.

3.4.3. `Panda Reacher`

We repeat the above point-reaching task using a 7 DoF Franka Emika Panda robot and 3D intermediate and goal positions. Since we do not care about the end-effector's rotation, the last DoF can be ignored, leading to an effective action dimension of 6. We use 8 basis functions per action dimension for the MPs, which yields a 48-dimensional problem. The expert demonstrations are collected using a teleoperation setup with a virtual twin that records joint values over time. Here, a human demonstrator moves a physical robot. The robot then sends its movement to a virtual twin and the joint values of the twin are recorded over time to generate expert demonstrations. Figure 3.3 shows an overview of the virtual part of this setup. We preprocess the human demonstrations by removing initial steps until the robot starts moving. We further repeat the last recorded time-step before fitting the ProMPs to ensure that the fit trajectories end near the goal position and do not over-smooth the human demonstrations.

The environment ranges from $(0, -0.5, 0)^T$ to $(1, 0.5, 1)^T$ meters. The targets are drawn randomly from uniform distributions with range 0.1 meters and means at $(0.5, 0.2, 0.6)^T$ and $(0.3, -0.1, 0.3)^T$ meters, respectively. A target is considered reached within a radius of 0.05 meters, and we consider a demonstration successful if both targets are reached within these radii. The robot starts all trajectories at its default resting position. We use the same geometric descriptors as for `Planar Reacher`.

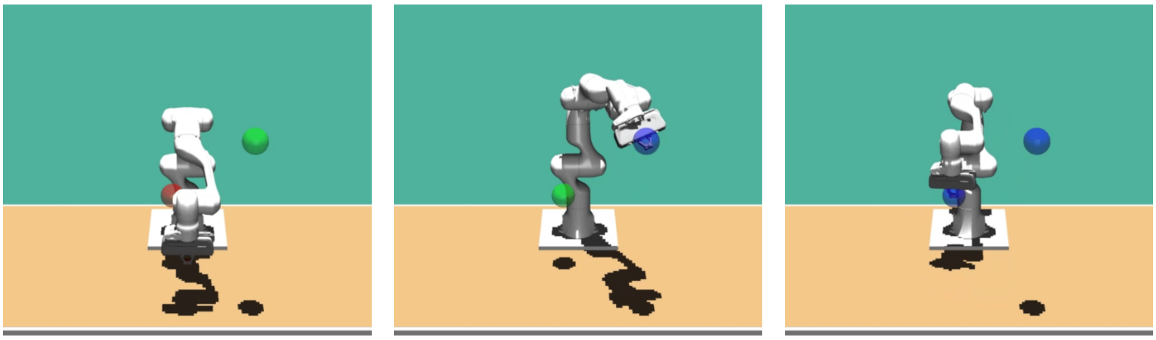


Figure 3.3.: Virtual part of the virtual twin setup for collecting human demonstrations for `Panda Reacher`. Starting from a resting position (**left**), the human demonstrator is tasked to reach the intermediate target (green). Once reached, both targets change color (**middle**). The task is successful if both targets have been reached (**right**).

3.4.4. `Box Pusher`

We also evaluate our approach on `Box Pusher`, a contact-rich robot manipulation task. A simulated Franka Emika Panda robot has a rod attached to its end-effector, which is used to push a rectangular box to a given goal position and orientation. Here, the methods need to account for the contact and the resulting mismatch between planned and executed trajectory. The box always starts at the same position, and the task context is given by the desired (x, y) translation of the box, plus its desired rotation in degrees. For simplicity, the task is learned in task space with a fixed height.

Demonstrations are collected using a virtual setup similar to that of `Planar Reacher`'s demonstrations. The demonstrator controls a mouse in the (x, y)-plane that the otherwise fixed end-effector of the robot follows using inverse kinematics. The trajectories, including the initial position of the end-effector, are fit in the (x, y) plane of the task-space for simplicity. To roll out a given trajectory on the robot, the (x, y) coordinates over time are concatenated with a fixed z-value and a downward rotation and given to an inverse kinematics controller. We sample the contexts from box translations and rotations that are feasible to demonstrate within a single smooth movement. As there tends to be a high correlation between the (x, y)-displacement and the rotation angle in the resulting contexts, we make sure that the training contexts are balanced with respect to this correlation. The demonstrations contain the starting position of the robot as well as its trajectory. The versatility in this task comes from different pushing techniques, such as touching the box from the inside or the outside.

The geometric descriptors consist of the (x, y) distances of the end-effector to the *initial* and *target* position of the corners of the box. To facilitate generalization between contexts, we translate the basis of these distances such that the target box aligns with the origin of the coordinate system. We additionally add the end-effector velocity and acceleration as well as the time-step, resulting in 19 features per time-step. This geometric description of the task allows for IL without simulating or explicitly modeling the box, significantly speeding up the training process and facilitating generalization to both new contexts and real-world robots. In order for the state-action baselines to be able to choose an initial position for the trajectory, we use a separate copy of the behavioral descriptors as input for the state-action at time-step 0. In other words, we duplicate the descriptors, using them once for step 0 and once for all other steps. We set the unused features to 0. The output of the policy for this first time-step is then interpreted as the starting position relative to the center of the box. Outputs in later steps correspond to velocities in the (x, y)-plane. We additionally conducted preliminary experiments with variants of *BC(S)* and *BC-GMM(S)* that computed distance features with respect to the current, online box position. We find that this choice of descriptors leads to diverging behavior on test contexts.

We use 3 demonstrations per training context and evaluate the average distance of the corners of the final box to that of the *target* position. A demonstration is considered successful if the average distance of the corners of the final box to that of the target position is less than 1.5cm, which roughly corresponds to the maximum error made by any expert demonstration.

3.4.5. Parameter Studies

We conduct extensive parameter studies to investigate how our design choices affect our approach for *VIGOR* and *EM+D-REX*. Concretely, we investigate the choice of geometric descriptors, directly training on sub-sampled human demonstrations without fitting a ProMP, and different discriminator architectures and sizes. We also evaluate the effects of different numbers of GMM components, using a discriminator ensemble and a different discriminator loss. For *EM+D-REX*, we look at the effect of the number of used EM components, and at algorithm-specific hyperparameters needed to tune the D-Rex reward. Section A.1 provides further detail.

3.5. Results

3.5.1. Quantitative Results

Figure 3.4 shows results on test contexts for `Planar Reacher` on the left and `Panda Reacher` on the right. We find that a trajectory-based setting and the use of mixture policies help performance, as can be seen from *VIGOR*, *EM+D-REX*, and *BC-GMM(T)*. Similarly, using a discriminator to iteratively optimize the policies is advantageous on the reaching tasks, as seen from *VIGOR* and *GAIL*. The clear distribution matching objective of *VIGOR* and *EM+D-REX* seems crucial for imitation from a low number of versatile demonstrations. *VIGOR* uniquely combines the above properties and reliably reaches both targets for both tasks.

Figure 3.5 shows the corresponding success rates on test contexts for both tasks. For both tasks, *VIGOR* consistently reaches both target circles, while all other methods except

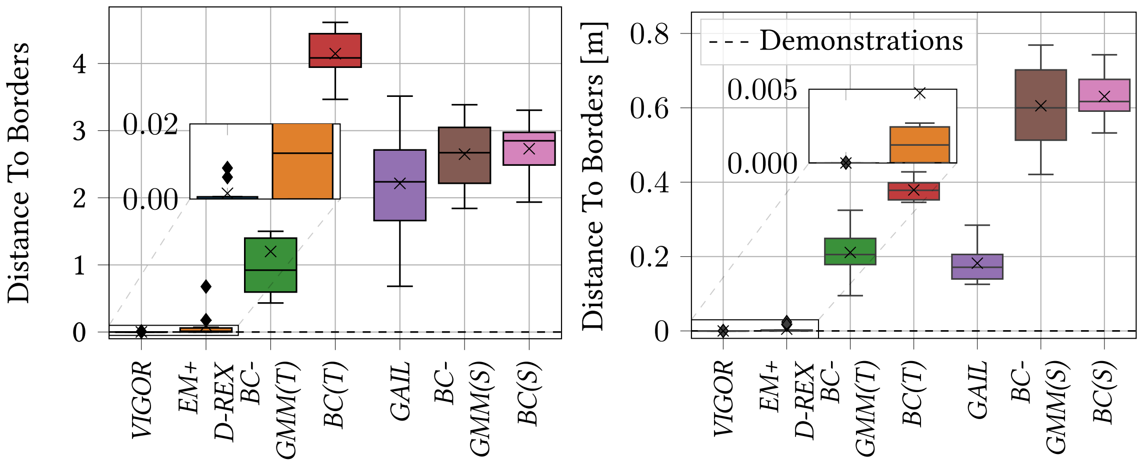


Figure 3.4.: Mean target distance of samples from the best policy component on `Planar Reacher` (**left**) and `Panda Reacher` (**right**) for test contexts. *VIGOR* and *EM+D-REX* consistently get close to both targets, while other methods fail to pursue at least one of the two targets. Yet, *EM+D-REX* seems to only represent a single mode and thus fails to reproduce the versatility of the demonstrations.

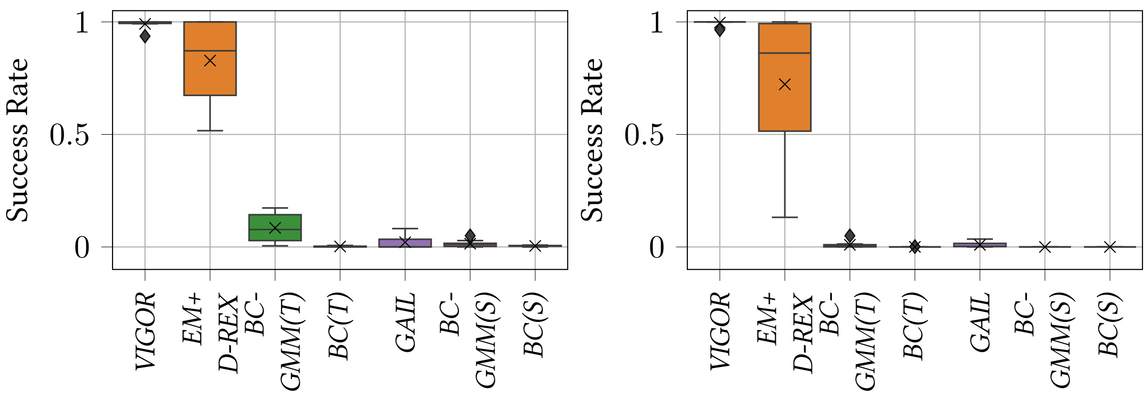


Figure 3.5.: (**Left**) Mean success rates on test contexts for the `Planar Reacher` task. (**Right**) Mean success rates on test contexts for the `Panda Reacher` task.

EM+D-REX struggle to do so. In general, the baselines with a GMM policy perform better than those without. *GAIL* performs the best of any state-action based approach.

Figure 3.6 shows `Box Pusher` results for training (left) and test (right) contexts. *VIGOR* reaches a target distance of almost zero on training contexts, suggesting that it can match the training distribution of behavioral descriptors of the human demonstrations. *EM+D-REX* performs poorly on the training set, presumably because the direct correspondence to the training data is lost by the intermediate reward representation. Most other baselines similarly improve over the initial box position, but do not come close to the human performance. This trend is more pronounced on test contexts, where only *VIGOR* reliably pushes the box to the desired target positions, while baselines fail to generalize to novel contexts, and in some cases show worse performance than the initial box configuration.

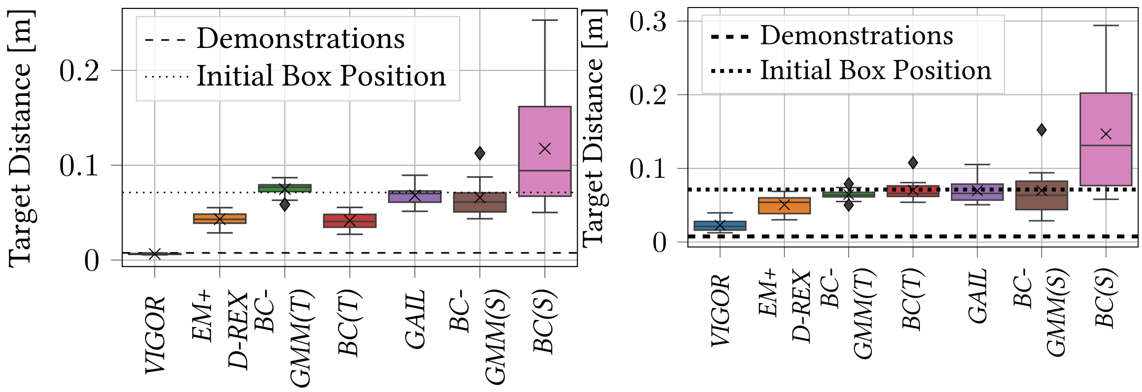


Figure 3.6.: Mean target distance of samples from the best policy component on `Box Pusher` for training (**left**) and test contexts (**right**). The dotted and dashed lines denote the average target distance of the initial position and the final position of human demonstrations, respectively. *VIGOR* consistently pushes the box to the right configuration in both training and test contexts, while baselines struggle to generalize from training to test contexts.

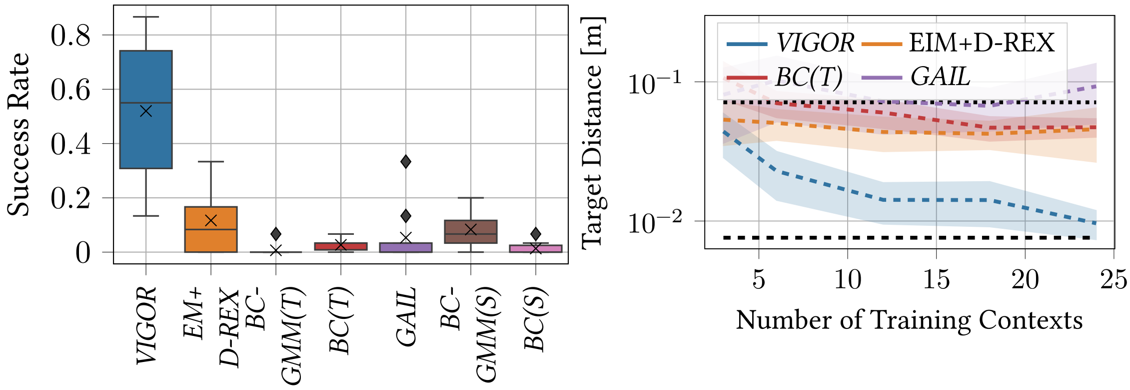


Figure 3.7.: (**Left**) Mean success rates on test contexts for `Box Pusher`. *VIGOR* shows the highest success by a wide margin. (**Right**) Mean target distance of samples from the best policy component on `Box Pusher` test contexts when varying the number of training contexts.

The left side of Figure 3.7 shows `Box Pusher` success rates on test contexts, which are strongly correlated with the target distances in Figure 3.6. *VIGOR* achieves the highest success rate, while *EM+D-REX* and *BC-GMM(S)* are sometimes able to solve the task. *BC-GMM(T)* does not produce successful solutions even though it improves the target distance, as seen on the left of Figure 3.6. Qualitatively, this behavior corresponds to trajectories that push the box in the right general direction, but do not account for a precise combination of target positions and angle. The right side of Figure 3.7 varies the number of training contexts and shows the resulting target distance on test contexts. *VIGOR* benefits most from additional training contexts, almost matching the demonstrator's performance on unseen contexts for 24 training contexts.

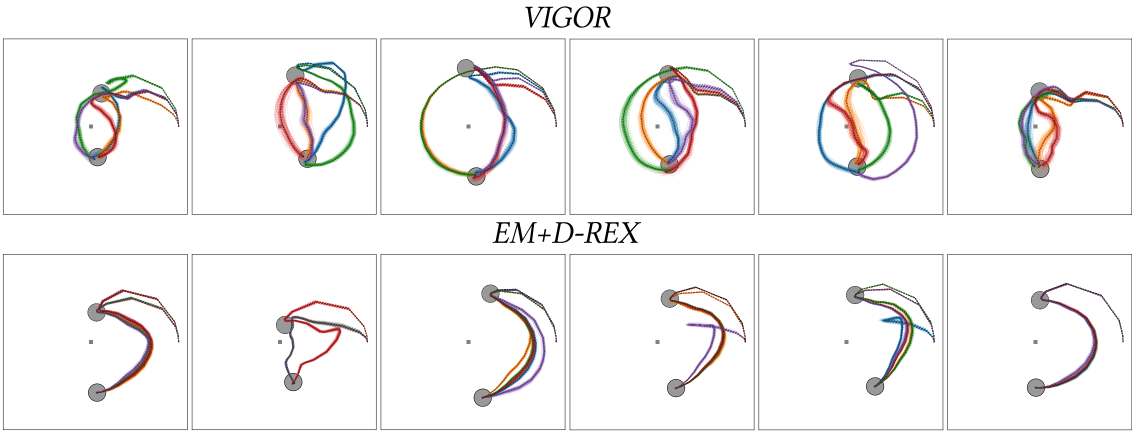


Figure 3.8.: Visualization of end-effector traces of `Planar Reacher` rollouts sampled from policies trained with *VIGOR* and *EM+D-REX*. Each component draws 100 samples and has its mean marked with a black border. *VIGOR*'s components specialize on different solutions, with each sample of a component being a feasible variation of a given type of solution. *EM+D-REX* components generally also reach the targets but do not capture the versatility in the behavior.

3.5.2. Qualitative Results

To explore the versatility of solutions learned by *VIGOR* and *EM+D-REX*, Figure 3.8 visualizes exemplary `Planar Reacher` policies learned by both methods. *VIGOR* produces versatile policies that closely match the demonstrations of the expert, with most components finding a different solution to the given task and samples of these components generally hitting both targets. In contrast, *EM+D-REX* policies often collapse to few different solutions, likely due to a lack of a clear distribution matching objective.

Figure 3.9 shows final states of the `Box Pusher` task for *VIGOR* for 6 test contexts for policies trained with demonstrations from 24 training contexts. Figure 3.10 shows the difference between desired and executed trajectories for 3 learned GMM component means on the same test context. The executed trajectories closely match the desired ones up to the contact with the box, where the force required to push the box changes the trajectory. However, since *VIGOR* learns from geometric descriptors of *planned* expert demonstrations, it compensates for this mismatch by design, allowing for offline training on planned trajectories that implicitly factor in contact with the box. Finally, we illustrate how the planned trajectories of *VIGOR* perform on a real robot. To this end, we create a real-world replica of the simulated box that the robot manipulates. Rollouts for different policy components of *VIGOR* trained on 24 training contexts on an exemplary test context are shown in Figure 3.11. We find that *VIGOR* straightforwardly generalizes from simulation to the real robot, likely due to its abstract geometric descriptors, offline trajectory generation, and the use of ProMPs. *VIGOR* successfully pushes the box to the desired target positions in different ways on the real robot. For example, the robot in the first row pushes the box from the inside, while the robot in the second row pushes it from the outside. The third trajectory also pushes from the inside but is much farther away from the corner of the box.

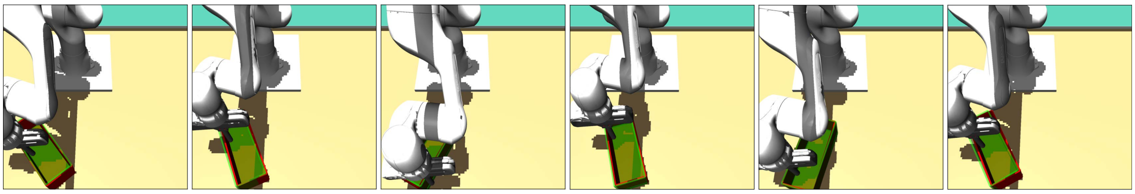


Figure 3.9.: Visualization of the final state of the `Box Pusher` task on 6 test contexts for *VIGOR* policies trained on 24 training contexts. The pushed boxes (red) need to align as closely as possible with the target box positions (green).

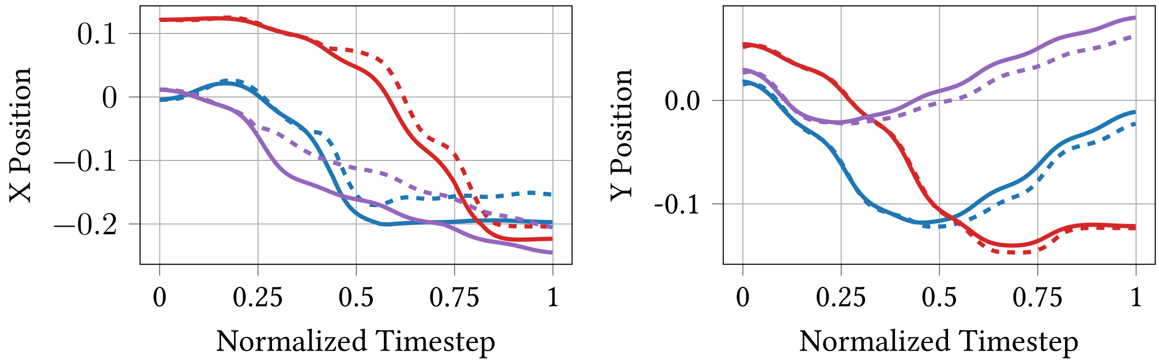


Figure 3.10.: Difference between planned (straight line) and executed (dotted line) trajectories for 3 learned GMM component means on the same `Box Pusher` test context. Contact with the box causes deviations from the planned trajectory, which *VIGOR* automatically compensates for due to its use of geometric descriptors and ProMP.

3.5.3. Parameter Studies

Table 3.1 shows the effect of different hyperparameters for *VIGOR* on `Planar Reacher`, sorting components by their performance. *VIGOR* benefits from concise behavioral descriptors and better matches ProMP fits of the human demonstrations rather than the demonstrations themselves. Additional data in form of more demonstrations and more training contexts aids performance. Similarly, additional mixture components and larger discriminators slightly improve the model. Section A.1 provides similar results for the `Panda Reacher` task, and validates training hyperparameter and reward choices for *EM+D-REX*.

3.6. Conclusion

We propose Versatile Imitation from Geometrically Observed Representations (*VIGOR*), a novel feature-matching approach to Imitation Learning in environments with versatile solutions. Utilizing a combination of movement primitives, mixture policies, and matching geometric behavioral descriptors, our method can closely imitate the behavior distribution of human experts from a small number of demonstrations. We show the effectiveness of our approach on a suite of challenging robot coordination tasks. The results demonstrate

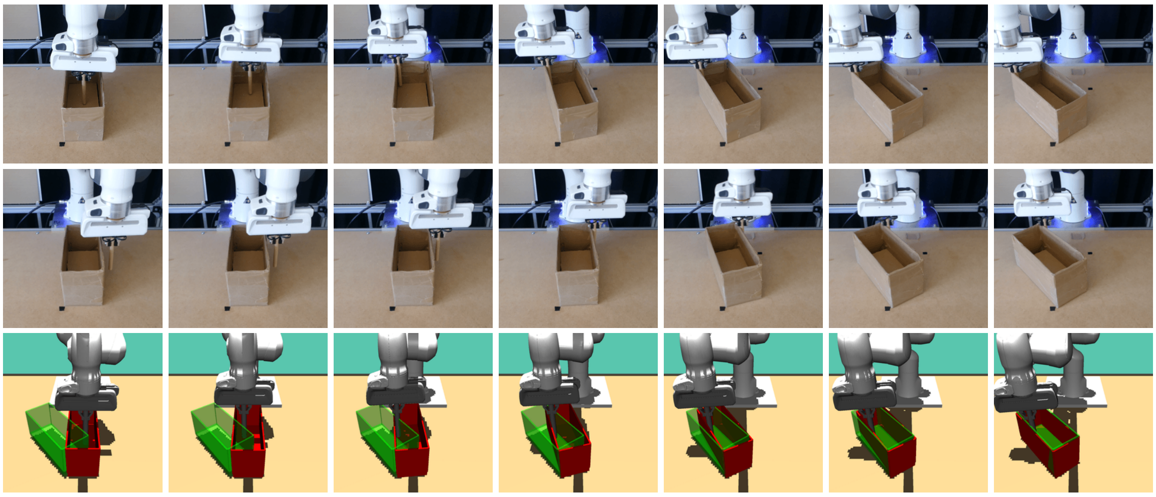


Figure 3.11.: Trajectory executions of three component means of a *VIGOR* policy trained on 24 training contexts for the same test context. The first two rows show a real Franka Panda executing the planned end-effector trajectories. The last row shows the same setup in the simulation, where the target box is overlaid in green. All three executions push the box in different ways, capturing the versatility of the human demonstrations.

that *VIGOR* is able to closely match the distribution of the demonstrator, outperforming the chosen baselines in all considered settings.

3.6.1. Limitations and Future Work

We currently train and maintain a separate GMM over ProMP parameters for each context, leading to linear space and time complexity with respect to the total number of contexts. One way to address this challenge would be to instead train a joint Mixture Density Network $q(\mathbf{w}|c)$ over all contexts, which is left for future work. Additionally, representing trajectories with a single ProMP limits them to single smooth movements. To alleviate this issue, MP *chaining* (Daniel et al., 2016; Neumann et al., 2009; Manschitz et al., 2014) can be straightforwardly integrated into our approach to allow for the representation of more complex movements.

VIGOR facilitates generalization to novel task configurations by using geometric descriptors. While this allows the user to include domain knowledge, it also requires a clear idea about which aspects of the task are important. Instead, geometric descriptors could be automatically extracted from the task for example by using neural networks processing point clouds (Qi et al., 2017a; Qi et al., 2017b).

Finally, our method assumes that the test configurations $\mathbf{c}_{\text{test}}$ are known in advance. Since the test configurations are optimized jointly with the training configurations, the only way to integrate new test configurations is to re-train the algorithm. In the future, we want to alleviate this issue by recovering a reward representation from ranked intermediate policies of the given training configurations, essentially combining our approach with some of the ideas of D-REX.

Parameter	Default	Value	Component				
			c_0	c_1	c_2	c_3	c_4
Reference			0.0015	0.0143	0.0221	0.0985	0.4269
Representation	Geometric	Angles	1.5262	2.1104	2.7000	3.4361	4.1781
		Jointwise	0.0115	0.0338	0.0825	0.2248	0.7324
Demonstrations	ProMP	Raw	0.1269	0.4263	0.8455	1.2909	2.1071
Architecture	1d-CNN	LSTM	0.0954	0.2373	0.3657	0.6142	0.9744
		MLP	0.1404	0.2177	0.3012	0.5363	0.9852
#Components	5	1	0.0667	–	–	–	–
		3	0.0029	0.0433	0.2997	–	–
#Discriminators	5	1	0.0032	0.0143	0.0849	0.1581	0.6240
Discriminator Loss	Stepwise BCE	BCE	0.0030	0.0084	0.0217	0.1038	0.3929
#Train Contexts	6	1	0.2590	0.3233	0.3767	0.4392	0.5683
		3	0.0002	0.0196	0.0536	0.1382	0.5001
		12	0.0000	0.0002	0.0024	0.0391	0.2328
#Demonstrations	5	10	0.0006	0.0019	0.0280	0.1100	0.3946
#Layers	2	1	0.0045	0.0312	0.0828	0.1472	0.6015
		3	0.0007	0.0053	0.0169	0.0802	0.3700
#Channels	32	16	0.0046	0.0282	0.0785	0.1699	0.5341
		64	0.0010	0.0020	0.0059	0.0431	0.4694

Table 3.1.: Parameter study of *VIGOR* on the `Planar Reacher` task. Green and red shading indicates improvement and degradation in performance relative to the reference configuration, respectively.

4. Adaptive Swarm Mesh Refinement using Deep Reinforcement Learning with Local Rewards

This chapter introduces Adaptive Swarm Mesh Refinement (*ASMR*), a novel approach to Adaptive Mesh Refinement (AMR) that treats the discretization process as a collaborative decision-making task. By framing each mesh element as an autonomous agent within a swarm, *ASMR* discovers accurate refinement strategies through direct interaction with a physical environment. This approach allows us to address our second research question.

Q2 How can we enable learning agents to leverage intrinsic geometric features to represent, reason over, and optimize the discretization of irregular physical domains?

ASMR utilizes on-policy Reinforcement Learning (RL), specifically Proximal Policy Optimization (PPO) as described in Section 2.1.2. To enable collaborative decision-making among the changing numbers of agents inherent to mesh refinement, we formalize a novel multi-agent RL framework. This framework utilizes local, agent-wise rewards derived from a fine-grained reference solution, making the approach applicable to a wide range of systems of equations and refinement criteria. We then construct a mapping of agents over time to distribute this local reward, ensuring that agents are rewarded for the long-term quality of the refinements they initiate.

To process local signals and features on the irregular and evolving mesh structure, we implement the policy via an Message Passing Network (MPN) architecture, following the principles detailed in Section 2.3.3. This architecture enables the agents to communicate and reason over the connectivity of the mesh in a permutation-invariant manner, allowing them to learn to coordinate their decisions based on local geometric and physical features. *ASMR* produces high-quality meshes with over 10 000 elements across several irregular domains and systems of equations without requiring explicit error indicators or other domain-specific heuristics.

The following work was published as **Swarm Reinforcement Learning for Adaptive Mesh Refinement** *(Niklas Freymuth, Philipp Dahlinger, Tobias Würth, Simon Reisch, Luise Kärger, Gerhard Neumann) in 36th Neural Information Processing Systems, NeurIPS 2023* and later extended to **Adaptive Swarm Mesh Refinement using Deep Reinforcement Learning with Local Rewards** *(Niklas Freymuth, Philipp Dahlinger, Tobias Würth, Simon Reisch, Luise Kärger, Gerhard Neumann), under Review, 2026. Reprinted with permission of the authors. Wording, notation, structure and formulations were revised in several places.*

4.1. Introduction

The numerical simulation of fundamental physical principles like mass, momentum, and energy conservation, often expressed through complex Partial Differential Equations (PDEs), is a cornerstone of modern engineering. Since analytical solutions of such PDEs are limited to simple cases, numerical approximations, particularly the Finite Element Method (FEM), are commonly employed (Brenner et al., 2008; Reddy, 2019; Anderson et al., 2021). The FEM provides a framework to find approximate solutions by transforming the continuous PDEs into a discrete system of equations. It partitions the continuous problem domain into a mesh consisting of smaller, finite elements, allowing for an efficient numerical solution whose accuracy depends on the number of used elements. However, as the physics becomes more complex, accurate simulations become significantly more expensive (Wanner et al., 1996; Zimmerling et al., 2022; Brandstetter et al., 2022).

To bypass the high computational costs of the FEM, an alternative line of research focuses on learned simulators that approximate physical solutions directly from data. Early work on these learned simulators uses simple feed-forward (Um et al., 2018; Zimmerling et al., 2019a) or CNNs (Guo et al., 2016; Zimmerling et al., 2019b; Zimmerling et al., 2022). Due to the used network architectures, this line of work is ill-suited for irregular, unstructured mesh-based representations in the FEM. Here, Graph Network Simulators (GNSs) (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2021) have emerged as an alternative architecture acting on a graph encoding of the simulation state. As an alternative to a data-driven learning approach, physics-informed neural networks (Raissi et al., 2019; Würth et al., 2023) directly optimize a neural network to predict simulations that satisfy the governing equations of a physical system. While usually mesh-free, recent extensions combine these approaches with GNSs and meshes to facilitate generalization to new domains during inference (Würth et al., 2024). All these learned simulators share a common goal of using recent advances in deep learning to speed up the simulation of complex physical systems. Yet, they do so by directly approximating the simulation, causing any prediction error of the learned model to directly affect the simulated quantities.

Instead of replacing the solver entirely, a more robust and risk-averse approach to speed up the FEM is Adaptive Mesh Refinement (AMR), which dynamically allocates more mesh elements to regions of high solution variability, striking a favorable balance between computational efficiency and accuracy (Plewa et al., 2005; Huang et al., 2010; Fidkowski et al., 2011). As AMR allows the creation of a simulation-specific mesh, it has become increasingly important for complex problems across domains like fluid dynamics (Berger et al., 1989; Baker, 1997; Zhang et al., 2020b; Wallwork et al., 2022), structural mechanics (Ortiz et al., 1991; Stein, 2007; Gibert et al., 2019), and astrophysics (Cunningham et al., 2009; Bryan et al., 2014; Guillet et al., 2019). Yet, general-purpose AMR methods often rely on either simple heuristics (Zienkiewicz et al., 1992) or computationally intensive error estimation techniques, such as goal-oriented adaptive finite element methods (Becker et al., 2023) and dual-weighted residual methods (Bangerth et al., 2013). While these approaches can achieve improved accuracy for specific applications, they are generally limited by their computational cost and lack of adaptability (Mukherjee, 1996; Kita et al., 2001; Yano et al., 2012; Cerveny et al., 2019; Wallwork, 2021), complicating their effective use in practical scenarios.

Several recent methods apply supervised learning to improve existing or devise new AMR strategies. Here, methods usually predict intermediary metrics for specific aspects of AMR, such as predicting an error or refinement marking per mesh element for a subsequent refinement step (Zhang et al., 2020b; Bohn et al., 2021; Roth et al., 2022; Wallwork et al., 2022; Służalec et al., 2023), or predicting local mesh densities over the considered domain (Huang et al., 2021). These supervised approaches typically propose a greedy and often local next-step refinement focused on minimizing the error in the next refinement step. However, this short-term focus may come at the cost of long-term optimization, as such refinements may not account for future regions of interest in time-dependent problems (Yang et al., 2023b), or distant regions in static problems with spatially propagating errors, such as advection problems.

Finally, an emerging body of work employs Reinforcement Learning (RL) (Sutton et al., 2018) to formulate AMR, and specifically *h-refinement* (Arnold et al., 2000; Stevenson, 2008), i.e., element sub-division and combination, as a sequential decision-making process. Multiple strategies for RL-AMR (RL-AMR) have been developed, all of which are motivated by the potential of RL to optimize non-differentiable, long-term rewards that are formulated to correspond to an error measure on the mesh. When considering h-refinement, the discrete mesh structure evolves at each step, resulting in a dynamic state space that requires the RL policy to handle a varying number of inputs and outputs. Some work circumvents this issue by learning a global, per-mesh quantity (Gillette et al., 2024; Pan et al., 2023), acting on local mesh elements (Huergo et al., 2024; Dzanic et al., 2024), or sampling a fixed number of mesh edges to refine per step (Wu et al., 2023). More general methods that consider the full mesh either do so by iteratively selecting an element to refine (Yang et al., 2023a) or training on individual elements' refinements and then inferring on the full mesh (Foucart et al., 2023). Another recent method uses a value decomposition network (Sunehag et al., 2017) to learn the credit assignment of the individual elements on the mesh quality by decomposing a shared Q-function (Yang et al., 2023b). All of these approaches only scale to simple problems, either due to an expensive inference process (Yang et al., 2023a), misaligned objectives and high variance in the state transitions (Foucart et al., 2023), or noisy reward signals during training (Yang et al., 2023b).

In this work, we instead formulate AMR via h-refinement as a Swarm RL (Šošić et al., 2017; Hüttenrauch et al., 2019) problem, extending existing frameworks to a shared observation space and per-agent, spatial rewards. Crucially, we allow for agents to split into new agents over time to model element sub-division in the refinement process, mapping a per-agent reward signal quantifying the reduction in simulation error over refinement steps to assign credits for individual agents throughout an episode. We use a Message Passing Networks (MPNs) (Sanchez-Gonzalez et al., 2020), a type of Graph Neural Network (GNN) (Scarselli et al., 2009; Bronstein et al., 2021), for our policy due to their effectiveness in learning physical simulations (Pfaff et al., 2021; Brandstetter et al., 2022). Our method, Adaptive Swarm Mesh Refinement++ (*ASMR++*), consistently produces highly efficient mesh refinements with thousands of elements and can be applied to arbitrary systems of equations. It further generalizes to previously unseen domains, system dynamics and material parameters when trained appropriately, allowing for robust and adaptive solutions across a wide range of complex scenarios.

The methodology presented here builds upon and refines previously published work on Adaptive Swarm Mesh Refinement (*ASMR*) (Freymuth et al., 2023) in several distinct ways. While the original *ASMR* policies produced meshes at a fixed granularity, we now introduce an adaptive element penalty. This scalar value is added to the reward function to penalize over-refinement and is provided to the policy as context, allowing a single model to generate meshes of varying densities. Additional extensions include an improved mapping between agents over time that acts as a regularizer and leads to better results, an improved network architecture and an improved reward formulation based on the reduction of the maximum local error of the mesh. While this reward formulation was already introduced and analyzed in the initial publications with inconclusive results, we now present a large scale study demonstrating its improved performance compared to the previously used reward. Throughout this chapter, we refer to the version incorporating these changes as *ASMR++*, and denote its predecessor as *ASMR*. However, since the current iteration represents the an updated version of the method that leaves its key concepts and contributions intact, we refer to the architecture collectively as *ASMR* in the broader context of this thesis. Experimentally, we find that the proposed changes within *ASMR++* substantially improve the method on more difficult environment setups. We further provide additional visualizations of our method, including a schematic of the agent mapping and a qualitative error comparison to a uniform mesh. Finally, we add two challenging environments that utilize Neumann boundary conditions and a 3-dimensional domain to showcase the applicability of our method to a wider range of applications. Figure 4.1 provides a schematic overview of a single refinement step, which is conceptually identical for *ASMR++* and *ASMR*.

We validate the effectiveness of *ASMR++* on a wide range of PDEs that require complex refinement strategies. Our experiments use simplicial meshes for simplicity, although *ASMR++* is agnostic to the element type of the underlying mesh. We employ conforming elements and corresponding h-adaptive refinements (Arnold et al., 2000; Stevenson, 2008), i.e., refinements via element subdivision, due to their prevalence in engineering (Ho-Le, 1988; Jones et al., 1997; Nagarajan et al., 2018). Compared to non-stationary settings from related work, which only consider a low maximum refinement depth or local refinements, the challenge here is to find multiple levels of precise refinement on the full mesh. We consider several recent RL-AMR methods as baselines (Yang et al., 2023a; Yang et al., 2023b; Foucart et al., 2023). As these methods have been shown to work for mesh refinement and coarsening with a relatively low depth on dynamic problems, we adapt them to our setting of stationary meshes that require multiple precise refinement steps and thousands of elements. We further compare to a threshold-based refinement heuristic that uses the popular Zienkiewicz-Zhu Error Estimator (ZZ Error) estimate (Zienkiewicz et al., 1992). Additionally, we consider different *oracle error estimates*, so called because they rely on

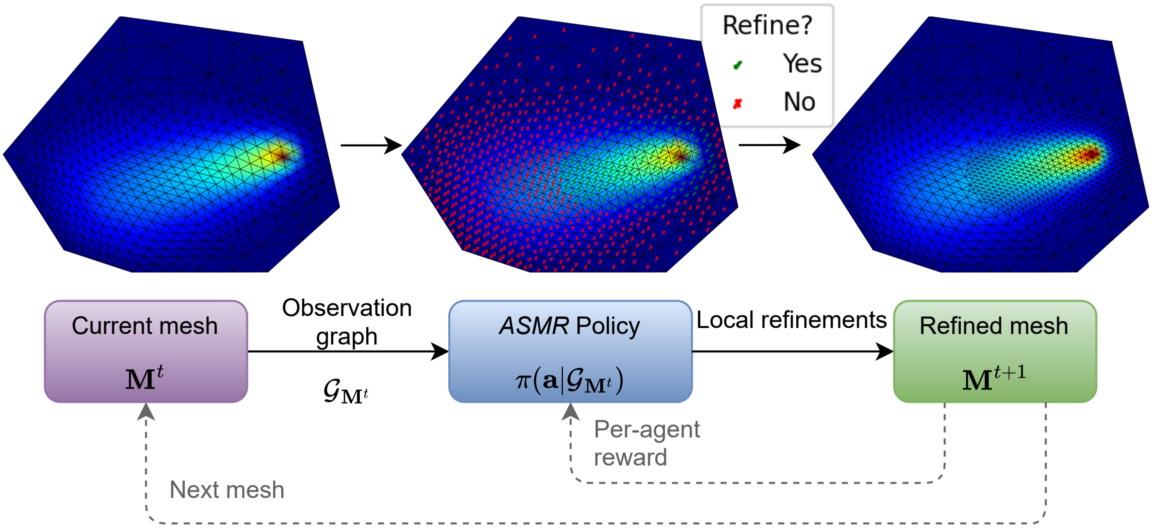


Figure 4.1.: A schematic refinement step for both *ASMR* and *ASMR++*. Given a current mesh, an observation graph encodes the elements as nodes and their neighborhood relationship as edges. A policy consumes this graph to decide on per-element markings that are given to a remesher, which subsequently produces a refined mesh. Based on the quality of this finer mesh, per-agent rewards are calculated. The process is repeated until the mesh is fully refined.

privileged information from a high-resolution uniform mesh to approximate the true solution error, to mark elements for refinement.

4.2. Adaptive Swarm Mesh Refinement++

ASMR++ treats elements of a mesh as a swarm of homogeneous agents that collaborate to find an optimal refinement. For this, each agent's state and observation are defined through its topological position in the mesh and includes local, rotation- and translation-invariant features of both the mesh and the system of equations. At each step, all agents decide whether to mark their respective element for refinement. Agents that refine their element receive a *local* reward based on how much their refinement has improved the quality of the underlying simulation. These local rewards are aggregated into a global term for the return to ensure that each mesh element optimizes its local region while contributing to the global solution quality. Crucially, each refinement subdivides the refined elements, introducing new agents in the process. We introduce a Adaptive Swarm Markov Decision Process (ASMDP) that features a mapping of agents over time, allowing us to optimize over multiple refinement steps by propagating reward from agents of later time steps back to related earlier agents. Figure 4.1 provides a schematic overview of *ASMR++*, and the following sections describe the individual aspects in more detail.

We implement all environments as OpenAI gym (Brockman et al., 2016) environments and publish our code, including all approaches and environments presented in this paper, at `https://github.com/NiklasFreymuth/ASMRplusplus`.

4.2.1. Adaptive Swarm Markov Decision Process

We view mesh refinement as a collaborative multi-agent RL problem with a changing number of homogeneous agents and agent-wise rewards. For this purpose, we adopt a swarm RL perspective by extending the SwarMDP framework (Šošić et al., 2017; Hüttenrauch et al., 2019) to accommodate vector-valued rewards and changing state, action, and observation spaces. This framework is itself an extension of the Markov Decision Process (MDP) introduced in Section 2.1.2. Formally, let an Adaptive Swarm Markov Decision Process (ASMDP) be a tuple $\langle \mathbb{S}, \mathbb{O}, \mathbb{A}, p, p_0, \mathbf{r}, \xi, \mathcal{K} \rangle$. The state space, observation space and action space are given by $\mathbb{S}$, $\mathbb{O}$, and $\mathbb{A}$ respectively. Since the number of agents changes over time, we denote subsets of the state, observation, and action spaces with N agents as $\mathcal{S}^N \subset \mathbb{S}$, $\mathcal{O}^N \subset \mathbb{O}$, $\mathcal{A}^N \subset \mathbb{A}$. The transition function $p : \mathcal{S}^N \times \mathcal{A}^N \to \mathcal{S}^M$ takes an action over a state of N agents and leads to a new state with M agents. The distribution over initial states is given as $p_0 \subseteq \mathbb{S}$. The reward function $\mathbf{r} : \mathcal{S}^N \times \mathcal{A}^N \to \mathbb{R}^N$ takes the full state into account to produce a scalar reward for each agent. Similarly, the observation function $\xi : \mathcal{S}^N \to \mathcal{O}^N$ calculates local observations for each agent from the global state.

We consider a finite-horizon setting with T time steps. In the scalar reward case, i.e., with $r(\mathbf{s}, \mathbf{a}) \in \mathbb{R}$, an optimal RL policy $\pi_\theta : \mathbb{O} \times \mathbb{A} \to [0, 1]$ with parameters θ generally maximizes the return, i.e., the expected cumulative future reward

$$J^t := \mathbb{E}_{\pi_\theta(\mathbf{a}|\xi(\mathbf{s}))} \left[\sum_{k=0}^{T} r(\mathbf{s}^{t+k}, \mathbf{a}^{t+k}) \right]. \tag{4.1}$$

A key challenge of using RL for AMR is the changing number of agents introduced by splitting agents that are marked to be refined. To adapt the SwarMDP framework and its underlying Markov Decision Process to changing numbers of agents within a single episode, we thus introduce an agent mapping $\mathcal{K}^{(t)} \in \mathbb{R}^{N \times M}$. Intuitively, each entry $\mathcal{K}_{ij}^{(t)}$ describes the responsibility of agent i at step t for agent j at step $t + 1$. The responsibilities of agents at step t for agents at step $t' > t$ can then be computed via the matrix multiplication

$$\mathcal{K}^{(t,t')} := \mathcal{K}^{(t)} \mathcal{K}^{(t+1)} \dots \mathcal{K}^{(t')} = \pi_{\theta}{}_{\tau=t}^{t'} \mathcal{K}^{(\tau)}.$$

Given this mapping, we can propagate the rewards of all future agents for which agent i is responsible for at step t to calculate a return

$$J_i^t := \mathbb{E}_{\pi_\theta(\mathbf{a}|\xi(\mathbf{s}))} \left[\sum_{t'=t}^{T} \gamma^{t'-t} (\mathcal{K}^{(t,t')} \mathbf{r}(\mathbf{s}^{t'}, \mathbf{a}^{t'}))_i \right]. \tag{4.2}$$

For training, e.g., a value function $V^{\pi_\theta}(\mathbf{s})$, this leads to a TD error (Sutton et al., 2018)

$$\delta^t = \mathbf{r}(\mathbf{s}^t, \mathbf{a}^t)_i + \sum_j \mathcal{K}_{ij}^{(t)} V_j^{\pi_\theta}(\mathbf{s}^{t+1}) - V_i^{\pi_\theta}(\mathbf{s}^t) \tag{4.3}$$

of actions $\mathbf{a} \sim \pi_\theta(\xi(\mathbf{s}))$. We derive the targets for Q-functions analogously.

The ASMDP framework allows for a collaborative optimization of changing numbers of agents, each of which is equipped with its own reward. Compared to, e.g., a centralized partially observable MDP (Yang et al., 2023a; Foucart et al., 2023), it provides a more efficient learning environment, since every environment step provides a learning signal for each agent. Similarly, the ASMDP supports a multi-agent setting with changing numbers of homogeneous agents, circumventing the posthumous credit assignment problem (Cohen et al., 2021) without having to resort to dummy states and learned value decompositions (Yang et al., 2023b). The agent mapping additionally allows for an optimization of the return, i.e., the reward over time, which would not be possible with a purely local per-agent view. While this work focuses on AMR, the ASMDP framework can readily be applied to other scenarios that are characterized by localized decisions of changing numbers of entities.

4.2.2. Actions and Agent Mappings

In the context of AMR, a system state $s \in \mathbb{S}$ is defined by the problem domain Ω, which we discretize as a mesh M, and a set of process conditions $\mathcal{P}$ comprised of a PDE, boundary and initial conditions. We refer to the latter as system of equations for simplicity. At step t, *ASMR++* views every element M_i^t of the mesh M^t as one of many homogeneous agents. Each agent has a binary action space, deciding on whether it wants to mark its element for refinement or not. These markings are provided to a remesher, yielding a finer mesh $M^{t+1} = \{Mj^{t+1}\}_j$. The remesher subdivides each refined element M_i^t into a set of smaller elements $\{M_j^{t+1}\}_j$, such that the disjoint union of all M_j^{t+1} reconstructs M_i^t, i.e., $\dot{\bigcup}_j M_j^{t+1} = M_i^t$. It may also refine unmarked elements to close the mesh and thus assert a conforming solution (Arnold et al., 2000), i.e., to make sure that elements of the mesh align with each other at the boundaries to ensure continuity of solution variables between adjacent elements.

ASMR uses an indicator function $\mathbb{I}(M_j^{t+1} \subseteq M_i^t)$ corresponding to the remeshing operation to map between elements and their successors. This function assigns each element to all elements that it subdivides into, or equivalently, lets each agent be responsible for all new agents that it spawns. We extend it to include a normalization factor $|M^t|/|M^{t+1}|$, yielding an agent mapping

$$\mathcal{K}_{ij}^{(t)} := \frac{|M^t|}{|M^{t+1}|}\mathbb{I}(M_j^{t+1} \subseteq M_i^t). \tag{4.4}$$

This formulation ensures $\sum \mathcal{K}^{(t)} = \sum \mathcal{K}^{(t')}$ for all t, t', which intuitively preserves the total 'mass' of agents over time. Empirically, the normalization factor regularizes the mapping of rewards within the agent swarm, akin to, e.g., batch normalization (Ioffe et al., 2015).

We can conceptually represent each mesh in a series of refinements as a layer in a hierarchical graph, with nodes corresponding to mesh elements. In our case, this graph is a simple tree. Each node connects to its predecessor in the prior layer and to all successors in the subsequent layer. Applying an agent mapping equates to navigating through the respective level of hierarchy of this graph. An example refinement procedure and its

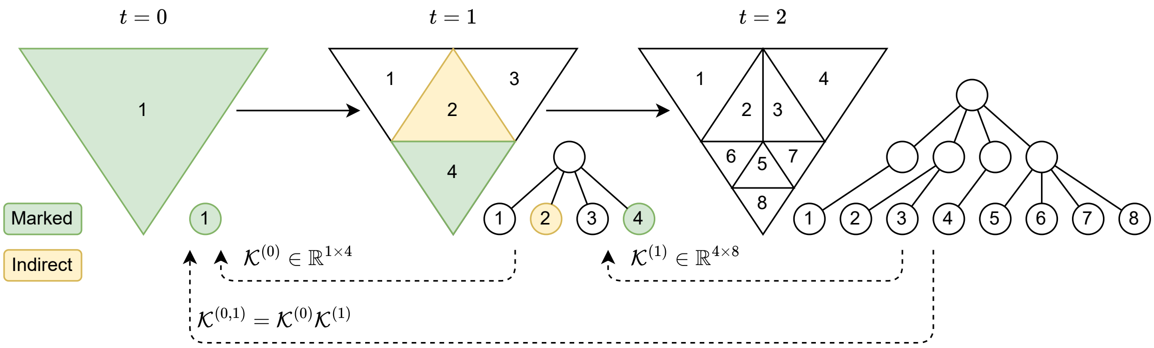


Figure 4.2.: Refinement procedure and responsibility mapping of *ASMR++*. (**Left**) An initial mesh has its single element marked for refinement. (**Middle**) The mesh element is subdivided into 4 new elements. The correspondence between the old and new mesh is represented as a directed acyclic graph. Constructing the matrix $\mathcal{K}^{(0)}$ from this graph allows us to map the new elements back to the old one. For the next step, Element 4 is marked for refinement, requiring an indirect refinement of Element 2 to ensure a conforming mesh. (**Right**) After the second refinement, the resulting mesh consists of 8 elements. Using the refinement graph to construct the mapping $\mathcal{K}^{(1)}$ allows us to compute responsibilities of elements in the previous mesh for this mesh. Chaining these mappings as $\mathcal{K}^{(0,1)} = \mathcal{K}^{(0)}\mathcal{K}^{(1)}$ allows us to directly compute responsibilities between the initial mesh and the mesh after two refinement steps.

corresponding refinement graph for an initial mesh with a single element and 2 refinement steps can be seen in Figure 4.2.

This work focuses on mesh (h-)refinement via iterative mesh subdivision. However, the above mapping can be extended to respect element coarsening as, e.g.,

$$\mathcal{K}_{ij}^{(t)} := \frac{|M^t|}{|M^{t+1}|}\left[\mathbb{I}(M_j^{t+1} \subseteq M_i^t) + \left(\mathbb{I}(M_i^t \subsetneq M_j^{t+1}) / \left(\sum_k \mathbb{I}(M_k^t \subsetneq M_j^{t+1})\right)\right)\right].$$

In this case, the hierarchical graph is no longer a tree but instead a directed acyclic graph, since multiple elements can coarsen into the same new element. Similar modifications can be made to adapt *ASMR++* to other AMR and Adaptive Mesh Generation (AMG) methods.

4.2.3. Observations and Policy Architecture

Given a mesh M^t and its corresponding system of equations, we construct the bidirectional graph $\mathcal{G}_{M^t} = (\mathcal{V}, \mathcal{E})$ with mesh elements as nodes $\mathcal{V}$ and their neighborhood relation as edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Nodes are initialized with features $\mathbf{h}_\mathcal{V}$ encoding the current system state, while edge features $\mathbf{h}_\mathcal{E}$ capture the relative geometric positioning of neighboring elements. Put together, this yields an observation $(\mathcal{G}_{M^t}, \mathbf{h}_\mathcal{V}, \mathbf{h}_\mathcal{E}) \in \mathcal{O}$, which we process using a Message Passing Network (MPN), as described in Section 2.3.3. The MPN outputs a learned representation $x_j = \text{MPN}(\mathcal{G}_{M^t}, \mathbf{h}_\mathcal{V}, \mathbf{h}_\mathcal{E})_j$ per node $v_j \in \mathcal{V}$. This learned representation can be fed into a policy head to compute an action per agent, yielding a joint action vector $\mathbf{a} \sim \pi_\theta(\mathcal{G}_{M^t}, \mathbf{h}_\mathcal{V}, \mathbf{h}_\mathcal{E}) = \pi_\theta(\mathcal{G}_{M^t})$, or into a value head to compute a per-element value function estimate. Both heads are parameterized as MLPs that are shared across all agents, facilitating efficient parallel processing and a shared agent parameterization. Since MPNs are permutation-equivariant by design (Bronstein et al., 2021), this architecture ensures that the policy is invariant to the global indexing of mesh elements, i.e., that the elements are only defined by their features and local connectivity in the mesh. We note recent trends in surrogate modeling for engineering applications (Alkin et al., 2024; Alkin et al., 2025) and RL for continuous control (Nauman et al., 2024b; Lee et al., 2024) increasingly favor heavily regularized large-scale architectures, such as transformers (Vaswani et al., 2017). While our MPN architecture offers several desirable inductive biases, these larger and more complicated policy backbones are a promising extension that may enable *ASMR++* to scale to larger domains and highly non-linear systems of equations.

4.2.4. Reward Definition

Adaptive Mesh Refinement (AMR) aims to generate meshes with enhanced local resolution in areas that are relevant for the underlying system of equations. Its objective is to achieve an optimal balance between the solution's accuracy on the mesh M^t, and its total element count $|M_i^t \in M^t|$. Since closed-form solutions are generally not available for most PDEs, we discretize the domain using a fine-grained uniform reference mesh M^* and use it to compute an approximate ground truth PDE solution u_{M^*} (Yang et al., 2023a). While calculating such a reference solution is expensive, it only needs to be computed once per training environment and is not required during inference. Given u_{M^*}, a refined mesh M^t and its solution u_{M^t}, we can define an *error per element* as some measure of difference between the solutions on this element. Based on this error estimate, we can then construct a reward function aimed at optimizing the trade-off between solution quality and element count. We propose two such error estimates and subsequent reward definitions below.

Maximum Reward. We can use the maximum difference over solutions as a measure of simulation quality that is independent of, e.g., the size of the used elements. Given reference elements M_m^* with midpoints $\mathbf{p}(M_m^*) \in M_m^*$ as sampling points to query the solutions on, this approach yields an error

$$\hat{\text{err}}(M_i^t) \approx \max_{M_m^* \subseteq M_i^t} \Big| u_{M^*}(\mathbf{p}(M_m^*)) - u_{M^t}(\mathbf{p}(M_m^*)) \Big|. \tag{4.5}$$

We efficiently calculate the assignment $M_m^* \in M_i^t$ required for this reward using a kD tree (Bentley, 1975). If a queried point $\mathbf{p}(M_m^*)$ exactly lies on the edge between two elements in M^t, which is possible for some refinement algorithms, we assign it to both of these elements with a weight of 0.5. We normalize the error estimate with the initial mesh error, yielding

$$\text{err}(M_i^t) = \hat{\text{err}}(M_i^t) / \sum_{M_j^0 \in M^0} \hat{\text{err}}(M_j^0),$$

which can be seen as a relative improvement of the mesh's elements. This normalization prevents systems of equations with large solution quantities and thus errors from dominating the learning process. From this error estimate, we then construct an element-wise local reward function that estimates the benefit of a refinement and compares it to the cost of adding additional elements. Using a scalar element penalty α, we define the local reward per element as the trade-off between mesh improvement and additional element cost, i.e.,

$$\mathbf{r}(M_i^t) := \left(\operatorname{err}(M_i^t) - \max_j \mathcal{K}_{ij}^{(t)} \operatorname{err}(M_j^{t+1})\right) - \alpha \left(\sum_j \mathbb{I}(M_j^{t+1} \subseteq M_i^t) - 1\right). \quad (4.6)$$

The local reward for each element is 0 if no refinement is made. Since Equation 4.6 is based on an element's decrease in maximum error, it is independent of the element size, ensuring that rewards of elements of different sizes are on the same scale without the need for explicit scaling. If an element is refined, its reward is positive if and only if the refinement decreases the highest error in this element by more than the scaled cost of adding new elements. This maximum error formulation directly targets reducing the worst-case error within each element. We find that this property results in well-behaved optimization, presumably since it directly targets refining elements where the reduction in error justifies adding new elements. *ASMR++* uses reward formulation by default in the following experiments.

Volume Reward. Equation 4.6 rewards the reduction in maximum element error for a given refinement. Alternatively, we can numerically integrate the error over each element's volume to compute an average error. Using the volume $\operatorname{Vol}(M_i)$ of the d-dimensional element M_i, we replace Equation 4.5 with

$$\widehat{\operatorname{err}}(M_i^t) \approx \sum_{M_m^* \subseteq M_i^t} \operatorname{Vol}(M_m^*) \left| u_{M^*}(\mathbf{p}(M_m^*)) - u_{M^t}(\mathbf{p}(M_m^*)) \right|.$$

We subsequently normalize this error as before and change the reward definition of Equation 4.6 to a volume-scaled reduction of this integrated error, i.e.,

$$\mathbf{r}(M_i^t) := \frac{1}{\operatorname{Vol}(M_i^t)} \left(\operatorname{err}(M_i^t) - \sum_j \mathbf{M}_{ij}^t \operatorname{err}(M_j^{t+1})\right) - \alpha \left(\sum_j \mathbf{M}_{ij}^t - 1\right). \quad (4.7)$$

The volume scaling component in Equation 4.7 encourages the policy to give priority to refining smaller mesh elements. This scaling roughly cancels out the volume of the integration points in Equation 4.5, resulting in a reward aimed at reducing the average error per unit volume. Optimizing this reward thus promotes a uniform distribution of error in relation to the size of each element across the mesh. This reward is used by the original version of *ASMR* (Freymuth et al., 2023), which we use as an additional baseline in our experiments.

Both reward formulations evaluate if, for any given refinement, the reduction in error is greater than the cost of adding additional elements. We compare both reward definitions in Section B.3 and show the difference in performance in Figure 4.21, finding that both variants perform similarly.

Multi-Objective Reward. In systems of equations with multiple quantities of interest, such as, e.g., elasticity problems where both the element displacement and stresses are

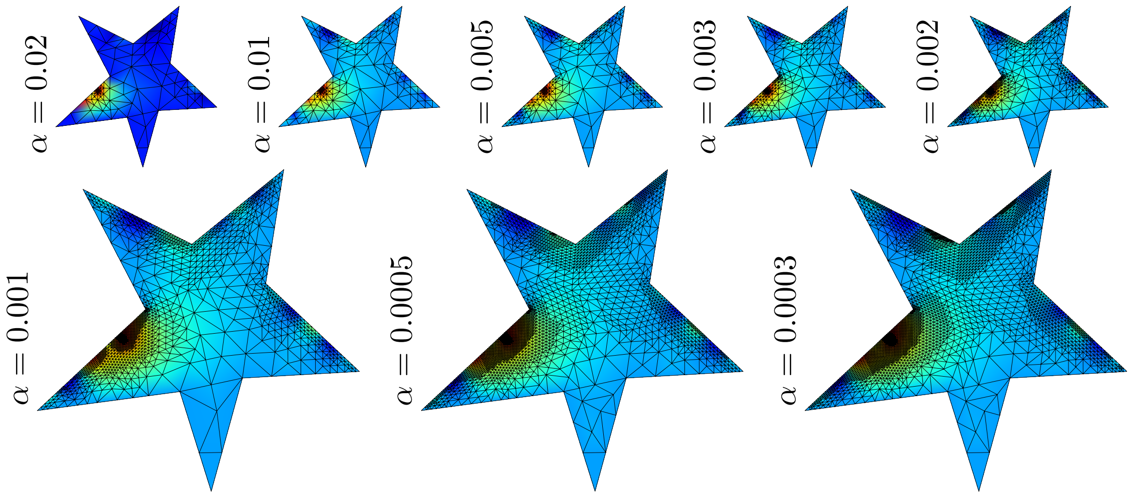


Figure 4.3.: *ASMR++* refinements for a Poisson problem with sinusoidal `Neumann Boundary` conditions for different inputs of the element penalty α. All refinements focus on the relevant parts of the problem, and lower element penalties lead to more fine-grained meshes.

important, the mesh should be optimized with respect to all the quantities of interest. Here, we simply calculate rewards independently for each solution dimension. We then apply some problem-dependent average or norm to compute a scalar reward per element.

Global Return. Equation 4.6 and Equation 4.7 reward each agent for its local refinement decision for each time step. While this direct and explicit local credit assignment leads to a temporally and spatially dense reward, it disregards non-local effects of mesh refinement, which are common for elliptical PDEs. To encourage global optimization, we therefore extend the return J_i^t of Equation 4.2 to include a global term as

$$J_i^{t\prime} = \frac{1}{2} J_i^t + \frac{1}{2} J^t, \tag{4.8}$$

where J^t is the global return of Equation 4.1 using the average reward $r = \frac{1}{N} \sum_j \mathbf{r}_j$.

4.2.5. Element Penalty

We add the element penalty α in Equation 4.6 an input to the policy to inform it about the penalty of adding new elements. During training, we draw α from a log-uniform distribution at the start of each episode, allowing for environment samples on a wide range of different penalties. Similar to Value Decomposition Graph Networks (VDGN) (Yang et al., 2023b), we then specify a value of α during inference to control the refinement level of the created mesh. Compared to *ASMR*, which trains a policy on a fixed element penalty value, we thus condition the policy on α for each rollout. This conditional policy allows for different mesh granularities during inference, whereas *ASMR* needs to train a new policy when the desired number of mesh elements changes.

Balancing the extra computational cost of adding mesh elements against the benefit of reducing simulation errors can be seen as an instance of multi-objective RL (Van Moffaert et al., 2014; Hayes et al., 2022). The cost of adding new elements increases linearly with a penalty factor α, and there exists some approximate ranking of the effectiveness of different mesh refinements at reducing simulation errors. This setup creates a roughly convex relationship between α and the policy's behavior. In other words, lowering α generally leads to a finer mesh that includes all refinements made with a higher α, while raising α results in fewer refinements throughout the mesh. Figure 4.3 provides an example of resulting *ASMR++* meshes for the same trained policy and `Neumann Boundary` environment for different values of the element penalty α. Regardless of the penalty, the parts of the mesh with the highest potential error reduction, in this case the areas near the load function and the sinusoidal Neumann boundaries, are refined the most.

4.2.6. Pseudocode

algorithm 2 and algorithm 3 provide pseudocode for the training and inference procedures of *ASMR++*. *ASMR++* iteratively interacts with a set of systems of equations during training, obtaining a reward by comparing the solution quality of its refined meshes to a fine-grained reference M^*. After training has finished, it can be applied to unseen problems without requiring access to such a fine-grained reference, providing accurate refinements from only the current coarse-mesh solution.

4.3. Experiments

4.3.1. Systems of Equations

We evaluate *ASMR++* on a diverse set of PDEs, using the Finite Element Method (FEM) as described in Section 2.3.1. Each PDE is defined by its weak formulation, a spatial domain, and initial and boundary conditions, which together specify a system of equations to be solved. The goal is to generate refined meshes that minimize the error of the FEM, relative to a fine-grained reference mesh M^* approximating the true PDE solution, while using as few mesh elements as possible.

Figure 4.4 shows examples for each environment and visualizes final meshes created using an *ASMR++* policy. Each environment is associated with a family of domains, including L-shapes, rectangles with a square hole or multiple rhomboid holes, convex polygons, star shapes with different numbers of peaks, and a 3-dimensional plate. From left to right and top to bottom, these environments are briefly characterized as follows. `Laplace`'s equation considers a rectangular heat source at the inner boundary and features high solution gradients near this source. `Poisson`'s equation uses a multi-modal load function comprised of a Gaussian Mixture Model (GMM), which in turn causes multiple distinct regions of interest in the solution. `Stokes Flow` solves for the velocity of a

Algorithm 2: *ASMR++* Training

Input: Training environment $\mathcal{E}$ (Sampler for PDE, M^0, M^*)
Input: Refinement steps T, Penalty range $[\alpha_{\text{min}}, \alpha_{\text{max}}]$
Input: Initial policy π_θ
Output: Optimized policy

while *not converged* **do**
  Data Generation:
  Initialize buffer $\mathcal{B} \leftarrow \{\}$
  for *each rollout* **do**
    Sample (PDE, $M^0, M^*) \sim \mathcal{E}$
    Sample penalty $\alpha \sim \text{LogUniform}(\alpha_{\text{min}}, \alpha_{\text{max}})$
    for $t = 0 \dots T-1$ **do**
      Get solution $\mathbf{u}_{M^t}$ from PDE and mesh M^t
      Construct graph $\mathcal{G}_{M^t}$ from $\mathbf{u}_{M^t}$ and M^t (Section 4.2.3)
      Sample refinement decisions $\mathbf{a}_t \sim \pi_\theta(\mathcal{G}_{M^t}, \alpha)$
      $M^{t+1}, \mathcal{K}_t \leftarrow \text{Refine}(M^t, \mathbf{a}_t)$
      Compute local rewards $\mathbf{r}_t$ using M^* and penalty α (e.g., Equation 4.6)
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\mathcal{G}_{M^t}, \mathbf{a}_t, \mathbf{r}_t, \mathcal{K}_t)\}$
    end
  end
  Policy Update:
  Compute global returns J^t (Equation 4.8) from rewards in $\mathcal{B}$
  Calculate loss and update parameters θ
end
return π_θ

Algorithm 3: *ASMR++* Inference (Deployment)

Input: Coarse mesh M^0, Trained policy π_θ
Input: Refinement steps T, Element penalty α
Output: Final refined mesh M^T

for $t = 0 \dots T-1$ **do**
  Get solution $\mathbf{u}_{M^t}$ from PDE and mesh M^t
  Construct graph $\mathcal{G}_{M^t}$ from $\mathbf{u}_{M^t}$ and M^t (Section 4.2.3)
  Select refinement decisions $\mathbf{a}_t \leftarrow \text{argmax}\ \pi_\theta(\mathcal{G}_{M^t}, \alpha)$ `// Deterministic`
  $M^{t+1}, \_ \leftarrow \text{Refine}(M^t, \mathbf{a}_t)$
end
return M^T

fluid streaming into the domain from inlet on the left side. It uses more complex shape functions and requires high precision near the corners of the rhomboid obstacles. `Linear Elasticity` simulates the deformation and the resulting stress of a metal plate. The `Heat Diffusion` environment simulates a time-dependent heat source and is evaluated at

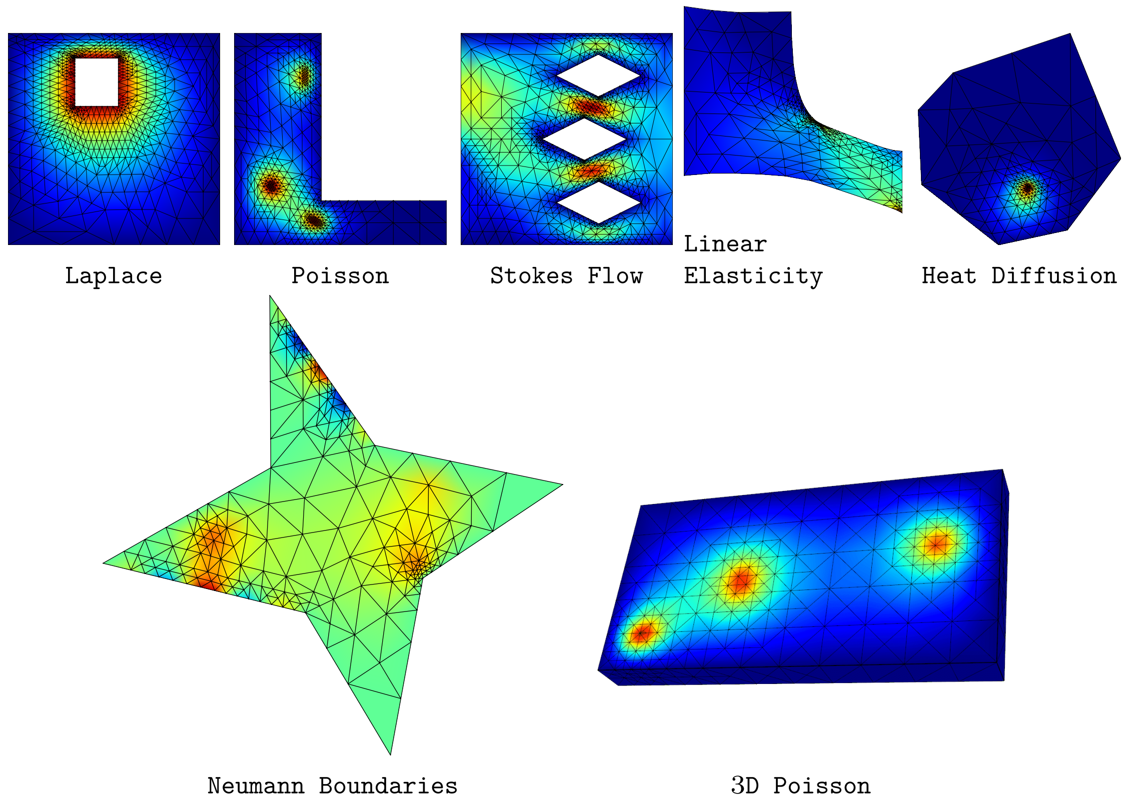


Figure 4.4.: Final *ASMR++* meshes and corresponding FEM solution for different environments. We visualize the velocity norm for `Stokes Flow` and a weighted average of the displacement norm and Von-Mises stress for `Linear Elasticity`. For the other environments, we visualize their scalar solution quantity as a heatmap. *ASMR++* provides highly adaptive meshes across a wide variety of PDEs by accurately optimizing environment-specific regions of interests. It additionally supports Neumann boundary conditions and volumetric 3d meshes.

the final simulation step. Adding sinusoidal `Neumann Boundaries` to Poisson's Equation models scenarios where the boundary flux is specified, resulting in high solution variability near the domain edges. Finally, `3D Poisson` considers a 3-dimensional variant of Poisson's equation, using tetrahedral meshes and a different underlying refinement algorithm.

We implement the FEM, all systems of equations, meshes and refinements in *scikit-fem* (Gustafsson et al., 2020). All environments are provided in the OpenAI gym (Brockman et al., 2016) standard. While *ASMR++* is designed to work with arbitrary element types, our experiments focus on conforming triangular and tetrahedral meshes. We use linear elements unless mentioned otherwise. Section B.1 details the systems of equations for all environments.

4.3.2. Adaptive Mesh Refinement

For each system of equations, we generate an initial coarse mesh M^0, which is then refined during an episode through iterative element subdivision. For simplicity, we use the default mesh refinement methods implemented in *scikit-fem* (Gustafsson et al., 2020). In 2-dimensional domains, we employ the red-green-blue refinement method (Carstensen, 2004), subdividing each marked element into 4 smaller elements and then closing the mesh, as illustrated in Figure 4.2. For 3-dimensional meshes, we apply longest edge bisection (Rivara, 1984; Suárez et al., 2005) to halve each marked element. We emphasize that *ASMR++* is compatible with arbitrary refinement algorithms, as it learns to mark elements for refinement based on the resulting error reduction.

In 2 dimensions, we uniformly refine the initial coarse mesh 6 times to create the reference mesh M^*, and use the same amount of 6 refinement steps per episode. We additionally evaluate a simpler setup with 4 refinements for the reference and AMR algorithm to emulate the problem complexity of existing work (Yang et al., 2023a; Fortunato et al., 2022; Yang et al., 2023b). In both cases, always refining all elements precisely reconstructs the reference mesh. In 3-dimensional domains, we uniformly refine three times, resulting in 8^3 times more elements in M^* compared to M^0 and employ 7 longest edge bisection AMR steps, yielding refinements that are topologically different from the reference mesh. The number of refinements was chosen to ensure that M^* is a sufficiently accurate proxy for the exact PDE solution to allow for challenging refinements, while remaining computationally feasible for efficient experimentation.

4.3.3. Training Setup

We train all RL-AMR methods for 10 random seeds on 100 randomly generated systems of equations, including randomized domains and process conditions. This choice reduces the number of expensive reference meshes M^* that must be computed during training, and is further explored in Section B.3. We use Proximal Policy Optimization (PPO) (Schulman et al., 2017) with discrete actions for the RL-AMR methods.

For all methods, we experiment with a range of target mesh resolutions to produce meshes of different refinement levels, as detailed in Section B.4.3. We repeat each experiment for 10 different random seeds, randomizing the parameters of the neural network as well as the PDEs, domains and process conditions. We normalize all domains to unit size unless mentioned otherwise. We create initial meshes with `meshpy` with a target element size of 0.05 for all environments except for `Neumann Boundary`, which uses a target element size of 0.02. For all experiments, we terminate an episode with a large negative reward of $-1\,000$ when a threshold of $20\,000$ elements, respectively $50\,000$ elements for `3d Poisson`, is exceeded.

Observation Graph. As node features for the observation graph described in Section 4.2.3, we use the mean and standard deviation of the FEM solution on the element's vertices, the element volume and the current environment timestep. For *ASMR++* and *VDGN-like*, we also add the current element penalty to condition the refinement process of the policy on a given target resolution. Some environments employ additional node features to encode

https://github.com/inducer/meshpy

their process conditions, which we describe in Section B.1. We use the Euclidean distance between the element midpoints as the single edge feature. Compared to using either a distance vector between element midpoints, or providing absolute positions, this ensures that the predicted sizing fields are invariant under Euclidean group operations (Pfaff et al., 2021; Bronstein et al., 2021).

Computational Budget. All experiments are run for up to 3 days on 8 cores of an Intel Xeon Platinum 8358 CPU, but usually terminate within a single day. We train a total of 5 RL-AMR methods on 7 separate environments for 10 repetitions each. Since 3 of these methods train on fixed number of target mesh elements, this leads to $(2 + (3 \cdot 10)) \cdot 7 \cdot 10 = 2\,240$ core experiments. A similar combined number of experiments is needed for the heuristics, parameter studies and preliminary experiments.

4.3.4. Evaluation

We evaluate the converged RL-AMR policies and heuristics on a fixed set of 100 randomly sampled systems, which are disjoint from the training data. We evaluate mesh quality by calculating the squared error between the solution on this mesh and the solution u_{M^*} on the fine-grained reference mesh M^*. This squared error over the domain is numerically approximated over evaluations at the midpoints $\mathbf{p}(M^*_m)$ of the reference mesh as

$$\sum_{M^*_m \in M^*} \text{Volume}(M^*_m)\left(u_{M^*}(\mathbf{p}(M^*_m)) - u_{M^t}(\mathbf{p}(M^*_m))\right)^2 . \tag{4.9}$$

We normalize this value by that of the initial mesh for comparability across PDEs, and call the resulting metric the remaining *squared error*. This squared error metric captures the overall error of the solution on the refined mesh, while punishing outliers. It differs from our optimization objective in Equation 4.6 and Equation 4.7, which instead reward minimizing the maximum or volume-adjusted average error of the solution on the refined mesh, respectively. Section B.2 discusses and evaluates a mean error and an approximation of the maximum error as alternate metrics. We list further algorithm and network hyperparameters in Section B.4.1.

Quantitatively, we measure the mesh quality using a Pareto plot of the number of elements and the normalized squared error of the mesh, as defined in Equation 4.9. We then calculate the interquartile mean (Agarwal et al., 2021) of the element counts and normalized errors of the 100 final produced meshes, and plot the resulting tuple. We repeat this process across a range of method-specific mesh resolution parameters, as described in Section B.4.3. For the RL-AMR methods, we plot the results across all 10 independently trained policy seeds. We further provide a log-log quadratic regression over the aggregated results of each method as a general trend-line. To improve clarity and focus on representative behavior, we omit data points with abnormally high element counts, which are sometimes produced as outliers by *VDGN-like*.

4.3.5. Baselines

RL-AMR Baselines. We compare to the initial version of *ASMR* (Freymuth et al., 2023), as well as to several recent non-stationary RL-AMR methods (Yang et al., 2023a; Yang et al., 2023b; Foucart et al., 2023). For *ASMR*, we use the setup and hyperparameters established in our previous work, including the reward definition of Equation 4.7, whereas *ASMR++* uses the improved reward of Equation 4.6. Compared to *ASMR++*, *ASMR* further omits the normalization factor in the agent mapping in Equation 4.4, considers an infinite horizon RL setting, and uses less regularization in the neural network architecture.

For the other RL-AMR methods, we use the rewards as described in the respective papers. *Single Agent* (Yang et al., 2023a) uses the difference in error norm as a global reward and marks a single element for refinement in each step via a categorical action over the full mesh. *Sweep* (Foucart et al., 2023) randomly samples a single mesh element and decides on its refinement based on local features and a global resource budget during training. The method uses the logarithm of the change in error between consecutive steps as a reward. During inference, each timestep considers all mesh elements in parallel. VDGN (Yang et al., 2023b) employs value decomposition networks (Sunehag et al., 2017) to estimate a global Q-function as the sum of element-wise local Q-functions. For a *VDGN-like* PPO variant, we instead analogously decompose the value function. We use an MPN policy for all RL-AMR methods except for *Sweep*, which utilizes a simple MLP.

Across all methods, we employ the absolute integrated difference in solution compared to the reference mesh, i.e., Equation 4.9 without the square as an error estimate. The RL-AMR methods handle the trade-off between mesh resolution and solution accuracy slightly differently. *Single Agent* simply refines for a number of steps, *Sweep* uses a target number of elements, and the other methods penalize each added element. Here, *ASMR* trains a new policy for each penalty, while only *ASMR++* and *VDGN-like* use a conditional policy that can produce meshes of different resolutions during inference. Section B.4.2 lists all baseline-specific hyperparameters, while Section B.4.3 provides details on the mesh resolution parameters for all methods.

Heuristic Baselines. In addition to the RL-AMR methods, we compare to a traditional error-based heuristics acting on a refinement threshold ϵ (Binev et al., 2004; Bangerth et al., 2012; Foucart et al., 2023). Given an error estimate $\text{err}(M_i^t)$ for element i, the heuristic marks all elements for refinement for which $\text{err}(M_i^t) > \epsilon \cdot \max_j \text{err}(M_j^t)$. We consider an *Oracle Error Heuristic* that assumes knowledge about the fine-grained reference solution u_{M^*} and uses the absolute integrated difference to this solution as its error metric. Similarly, we compare to a *Maximum Oracle Error Heuristic*, for which the error estimate of Equation 4.5 is used. Both variants require the reference mesh M^* to estimate an error, which is expensive to compute and thus may be unavailable during inference. We additionally consider the commonly used ZZ Error, which instead uses a superconvergent patch recovery process for its error estimate (Zienkiewicz et al., 1992) and does not require access to M^*. As the recovery process averages over neighboring mesh elements, the resulting *ZZ Error Error Heuristic* generally produces smooth error estimates. This property can lead to more coherent refinements when compared to the oracle heuristics, especially when considering the closing operations needed to ensure a conforming mesh after every refinement step. However, we find that the *ZZ Error Error Heuristic* error estimates are often inaccurate for coarse initial meshes due to insufficient gradient information between neighboring elements. We alleviate this issue by pre-refining the initial mesh twice before applying the *ZZ Error Error Heuristic*, which improves its performance significantly.

All heuristics greedily refine based on local error estimates, in the sense that they are based purely on the current error rather than the potential reduction in error. Thus, they implicitly assume that refining elements with high error is the best way to minimize the error on the mesh. This assumption does not always hold, as, e.g., elliptical PDEs often feature globally propagating errors (Strauss, 2007; Foucart et al., 2023). The RL-AMR methods instead learn to optimize an objective function, which rewards the error reduction over time. This difference between greedily marking elements with a high local error, and learning to anticipate the long-term error reduction for any given refinement allows the RL-AMR methods to find more global strategies that take multiple refinement steps and non-local information of the mesh elements into account.

4.3.6. Generalization Experiments

We experiment with the generalization capabilities of *ASMR++* in different ways. Generalization is crucial for practical scenarios, as it allows applying policies trained on narrow, inexpensive domains to a wider range of complex problems where training is prohibitively expensive. Specifically, we evaluate generalization across novel geometries and load functions for a fixed system, and the robustness of the policy when faced with varying material parameters across different physical systems. We additionally consider the ability to up-scale to larger domains during inference, allowing the use of small, efficient training domains. For training and evaluation, these generalization experiments follow the methodology and setup of the main experiments unless mentioned otherwise.

Different Domains and Load Functions. For the first setting, we experiment on the `Poisson` environment to test the generalization capabilities of our approach to novel domains and load functions. We consider 5 different 2D domain classes that are used for different environments, plus a rectangular domain $\Omega = (0,1)^2$. For each class, we randomly sample 3 domains and apply a GMM load function with 1, 3, and 5 components, respectively. This experiment serves to verify whether *ASMR++* can generalize to different domains and load functions during inference.

Different Materials. We then test the ability of *ASMR* to generalize across a wide range of systems of equations by introducing varying material parameters to `Heat Diffusion` and `Linear Elasticity`. We define three levels of variability for both environments, namely *fixed material* that keep the constant parameters of the main experiments, *small variation* were the material parameters are varied within a narrow range, and *large variation* where the parameters span a wide range. Material parameters are sampled log-uniformly for diffusivity and Young's modulus, and uniformly for the Poisson ratio. The log of the diffusivity is added as a node-level input feature to the heat diffusion *ASMR++* policy, while the log of Young's modulus and the Poisson ratio are added for linear elasticity. Table 4.1 summarizes the specific parameter ranges for each environment and variation level.

Table 4.1.: Parameter ranges for the material generalization experiments by environment. Parameter units are specified in parentheses. Meshes are normalized to unit length, so physical units correspond to this scale.

Environment	Parameter (Unit)	**Level**	**Value**
`Heat Diffusion`	Diffusivity (m^2/s)	*fixed material*	$1.0e{-3}$
		small variation	$[1.0e{-4}, 1.0e{-2}]$
		large variation	$[1.0e{-5}, 1.0e{-1}]$
`Linear Elasticity`	Poisson Ratio	*fixed material*	0.3
		small variation	$[0.2, 0.4]$
		large variation	$[0.0, 0.499]$
	Young's Modulus (GPa)	*fixed material*	1.0
		small variation	$[0.3, 3.0]$
		large variation	$[0.1, 10.0]$

Larger Domains. Finally, we experiment with the ability of *ASMR* to up-scale to larger domains during inference. Using `Poisson` as an example, we augment the training PDEs to cover a larger space of potential simulations and mimic patches of larger mesh segments by altering boundary conditions and load functions. We train our policies on the square hole domains of, e.g., the top left of Figure 4.4. We sample the means of the load function from a centered Gaussian with a standard deviation of 0.2, allowing components outside the mesh, and additionally apply random Gaussian loads to selected boundary edges to emulate a larger domain outside of the simulated mesh. As these changes lead to training PDEs of varying complexity, we use $10 \cdot \hat{\text{err}}(M)$ instead of $\text{err}(M)$ in Equation 4.6, i.e., we omit the per-domain normalization, and use an unlimited number of training PDEs to further increase variety. These changes only affect the training environments, and do not interfere with our training algorithm or inference scheme. Figure 4.5 visualizes exemplary training PDEs and policy refinements.

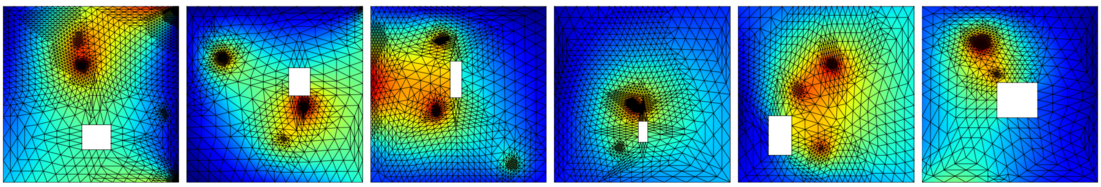


Figure 4.5.: Random augmented training PDEs and *ASMR++* refinements for the domain size up-scaling experiments on `Poisson`.

Once trained, we evaluate the policy on spiral-shaped domains of increasing sizes, ensuring the volume of the initial mesh elements remains constant. To align the complexity of the load function with the enlarged mesh sizes, we increase the number of GMM modes. Specifically, we use 16 components for larger domains, respectively 100 for the largest 20×20 domains. These components are first placed on a uniform grid and then randomly perturbed, allowing for them to be positioned outside of the mesh.

4.4. Results

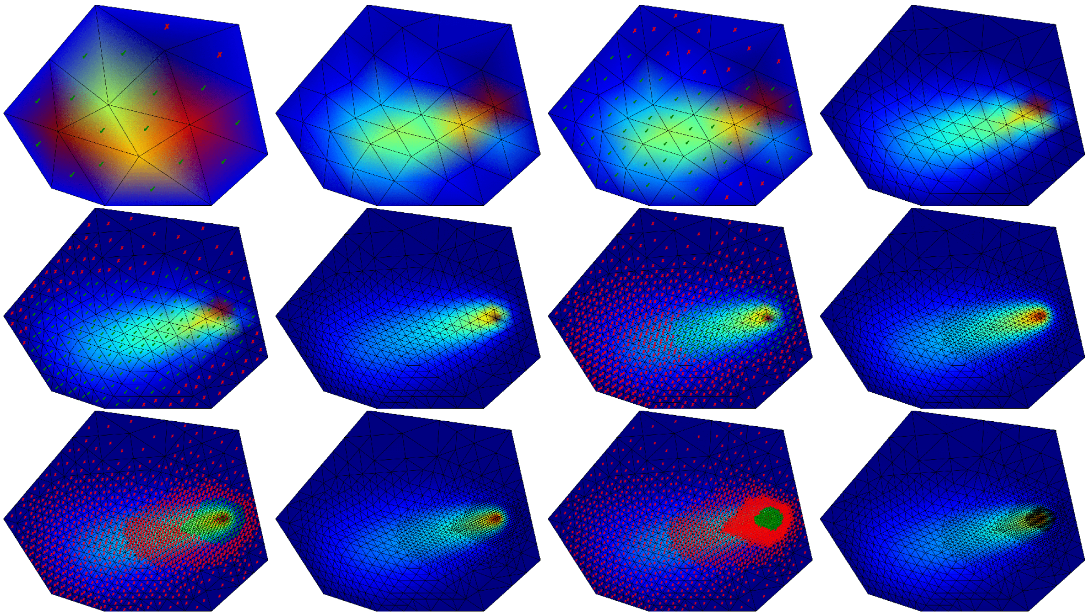
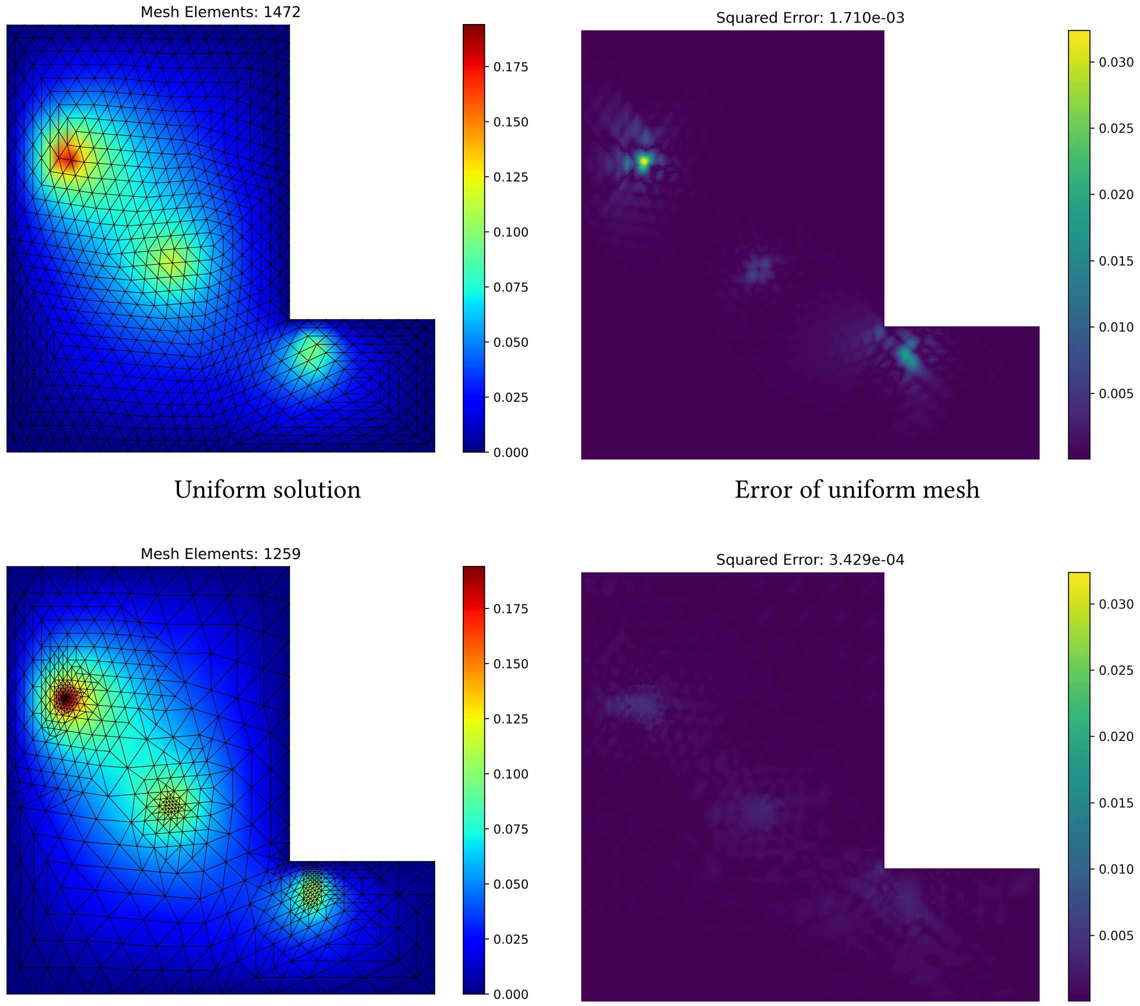


Figure 4.6.: Full *ASMR++* rollout for a heat diffusion example, including the policy action that marks individual elements for refinement. The mesh's background color denotes solution magnitude. The odd columns show which elements the policy marks for refinement (designated by a green tick), while the even columns show the resulting refined meshes. *ASMR++* produces a sequence of refinements that improves solution accuracy while using few total mesh elements.

4.4.1. Qualitative Results

Figure 4.6 visualizes a full rollout of *ASMR++* on an instance of `Heat Diffusion`, including the per-element refinement markings after every policy step. For this and other qualitative figures, the background color represents the magnitude of the problem-dependent scalar solution. *ASMR++* is trained to produce a sequence of refinements that improve the solution accuracy enough to justify the additional mesh elements, which here corresponds to more refinements along the path of the heat source. Figure 4.3 shows final refined meshes for the same `Poisson` problem for a fixed *ASMR++* policy that is conditioned on different values of the element penalty α. Here, decreasing the element penalty results in more mesh elements and thus a more accurate solution. Our method is able to focus on the interesting parts of the mesh regardless of the final mesh resolution, thus providing a favorable trade-off between solution accuracy and computational cost. Figure 4.4 provides final refinements of *ASMR++* on exemplary systems of equations for all considered environments. The refinement strategy individually adapts to each environment, choosing to refine elements that decrease the simulation error regardless of the underlying system of equations. We further explore this error reduction property in Figure 4.7. The visualization shows that our method achieves a much more uniform distribution of errors across the mesh when compared to simple uniform refinement, while also acquiring a much smaller simulation error for the same number of mesh elements.



Uniform solution | Error of uniform mesh

ASMR++ solution | Error for *ASMR++* mesh

Figure 4.7.: Comparison of exemplary *ASMR++* refinement with a uniform mesh on `Poisson`. (**Left**) Mesh and PDE solution for a uniform mesh (**top**) and our approach (**bottom**). Both meshes have a similar number of total elements, but different spatial resolutions. (**Right**) Normalized squared difference in solution to the ground truth PDE solution u_{M^*}, evaluated at each reference element's midpoint. *ASMR++* has less than a third of the error of the uniform mesh, and a much more even distribution of errors across the mesh.

Section B.5 visualizes all methods on `Stokes Flow` to showcase the different refinement behaviors for different target mesh resolution parameters. Section B.6 provides additional *ASMR++* visualizations for the remaining environments, including different camera angles for `3D Poisson` that uses tetrahedral meshes.

4.4.2. Quantitative Results

Figure 4.8 provides results for `Laplace`.

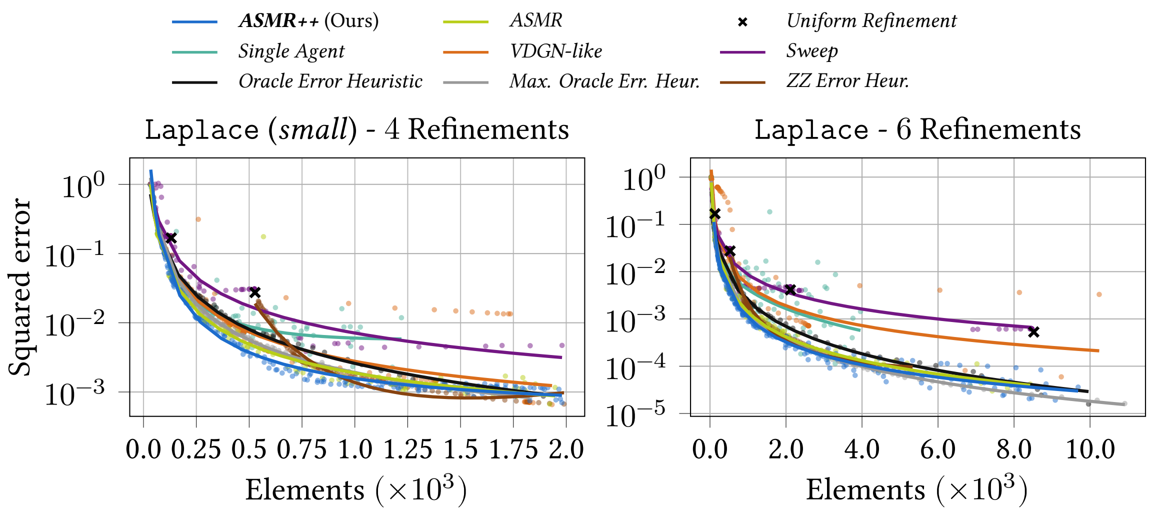


Figure 4.8.: Pareto plot of normalized squared errors and number of final mesh elements on `Laplace`. (**Left**) All RL-AMR methods produce better-than-uniform meshes for shallow refinements and reference meshes with only 4 refinement steps. (**Right**) When increasing the complexity to 6 refinement steps, most RL-AMR methods exhibit high variance in mesh quality and sometimes fail to provide useful meshes. *ASMR++* consistently produces high-quality meshes for both environment complexities, performing on par with or better than the heuristics and providing a slight but consistent benefit over *ASMR*.

We evaluate the mesh quality of all methods using a Pareto plot over aggregated squared error and number of mesh elements, as described in Section 4.3.4. On its left, we evaluate a simplified setup that uses 4 refinement steps for both the reference mesh and the AMR algorithms, except for *Single Agent*, which instead refines for up to 100 time steps. All methods clearly outperform uniform refinements. The *ZZ Error Heuristic* performs well for larger meshes, but can not produce meshes with few elements as it starts with 2 uniform initial refinements.

The right of Figure 4.8 increases the complexity to 6 refinement steps, and 400 time steps for *Single Agent*. Similarly, Figure 4.9 provides results on the remaining 6 environments. Across all environments, only *ASMR* and *ASMR++* can handle larger meshes while the other RL-AMR methods fail to provide better-than-uniform meshes or exhibit high variance in the quality of the proposed refinements. Our method additionally outperforms all heuristics, including the *Oracle Error* and *Maximum Oracle Error Heuristics*, in several instances. These results indicate that our optimization method facilitates learning non-local long-horizon refinement strategies that minimize the mesh error instead of simply targeting elements that currently have a high error. *ASMR++* further improves over *ASMR* in most settings, especially in the complex `Stokes Flow` environment where it compares well to the different *Heuristics*. In environments where *ASMR++* does not perform better than *ASMR*, both perform similar to or better than all heuristics, potentially indicating that both methods offer near-optimal refinements for these environments. On the `3D Poisson`, the different methods quickly converge to a relatively high final mesh error, likely due to the difference in mesh topology between the refined meshes and the uniform reference that is caused by the longest edge bisection refinement strategy.

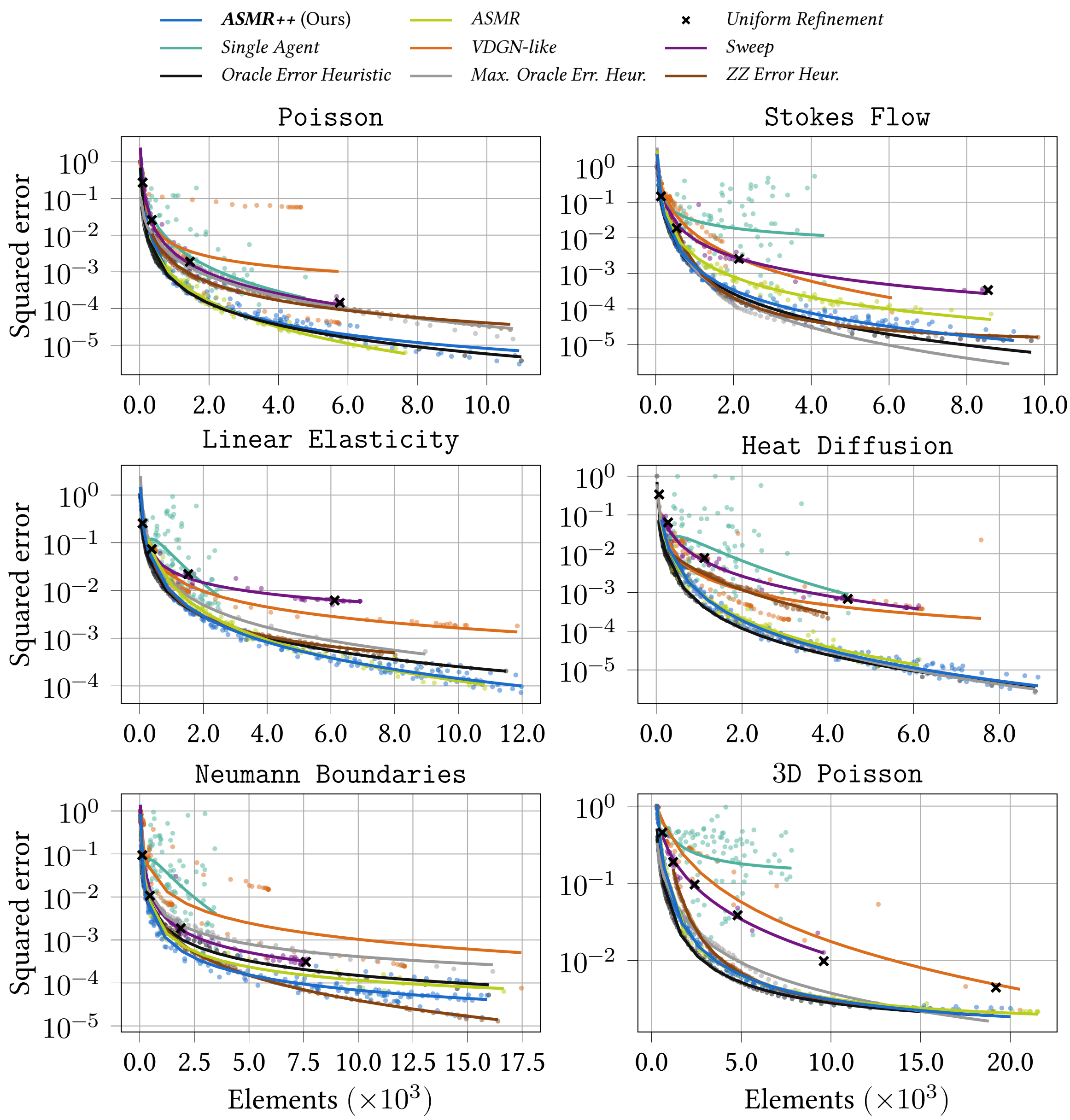


Figure 4.9.: Pareto plot of normalized squared errors and number of final mesh elements across different environments. On all environments, the error decreases in a log-linear relation to the number of used elements. *ASMR++* gracefully scales to meshes of more than ten thousand elements, significantly outperforming the learned *Single Agent*, *VDGN-like* and *Sweep* in terms of consistency and mesh quality. *ASMR++* also slightly improves over *ASMR* on most environments and generally compares favorably to the *Oracle*, *Maximum Oracle* and *ZZ Error Heuristics*.

Section B.2 presents additional results using a mean error and an adapted maximum error metric. On both metrics, *ASMR++* is generally competitive with or better than the *Heuristics* and significantly outperforms the *Single Agent*, *Sweep* and *VDGN-like* baselines. Comparing the different metrics, the *Maximum Oracle Error Heuristic* improves over the *Oracle Error Heuristic* on the maximum error metric and vice versa. Likewise, *ASMR*, which uses a scaled variant of the mean error instead of the maximum objective of Equation 4.6, outperforms *ASMR++* on half of the environments in terms of the remaining mean error, while *ASMR++* is superior for every setup when evaluating the maximum error metric.

4.4.3. Runtime Comparison

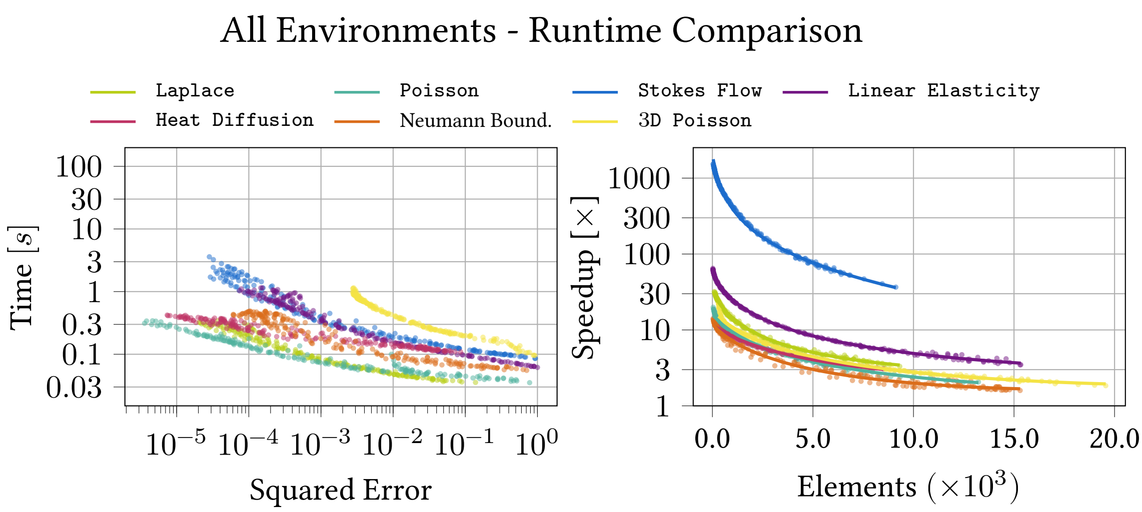


Figure 4.10.: Wallclock-time evaluation of *ASMR++* for all environments. Each color corresponds to a different environment. (**Left**) Runtime of *ASMR++* and the squared error when compared to the uniform solution u_{M^*}, which has an error of 0 by definition. (**Right**) Speed-up of *ASMR++* compared to calculating the uniform solution u_{M^*}, which contains roughly 10^5 elements, with specific numbers varying depending on the domain. Our approach is able to quickly produce high-quality meshes, achieving speedups of 1 to 2 orders of magnitude, depending on the environments and target resolution of the mesh.

Figure 4.10 shows the runtime comparison between our method and direct computation of the uniform reference M^* across all environments. Our method's timing includes the creation of the initial mesh, and iteratively solving the system of equations on the mesh, creating an observation graph, querying the policy for a refinement strategy, and finally refining a total of 6 times. For the uniform mesh, we simply measure the time it takes to refine the initial mesh 6 times and to subsequently solve the problem on the resulting mesh. We use a single 8-Core AMD Ryzen 7 3700X Processor for all measurements.

Our approach is always significantly faster than computing the fine-grained reference solution u_{M^*} despite the computational overhead of computing the observation graph and executing the policy. Notably, higher squared error tolerances lead to reduced computation time, showing that *ASMR++* can adapt to environment-specific computational budgets. `Stokes Flow` uses comparatively expensive P_2/P_1 Taylor-Hood-elements (John et al., 2016), causing our method to be more than 30 times faster than computing u_{M^*} even for highly refined and accurate final meshes.

The other RL-AMR methods feature a similar iterative refinement procedure and thus runtime, but produce meshes of worse quality than *ASMR++*. In contrast, the *Oracle Error Heuristic* and *Maximum Oracle Error Heuristic* produce high-quality meshes, but require computing u_{M^*} for an error estimate. As such, these heuristics are impractical in terms of runtime, as they depend on the fine-grained solution that AMR aims to approximate.

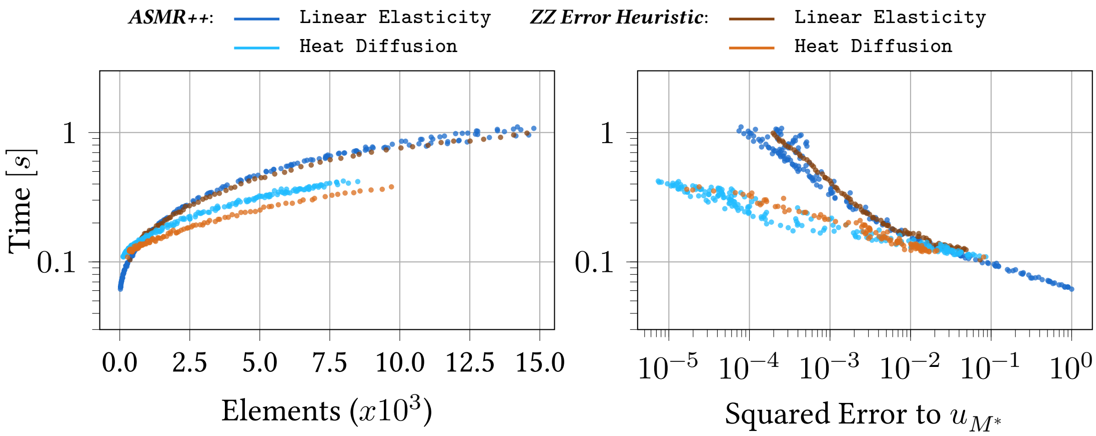


Figure 4.11.: Wallclock time comparison of *ASMR++* and the *ZZ Error Heuristic* for `Linear Elasticity` and `Heat Diffusion`. For the *ZZ Error Heuristic*, we measure the time required to solve the system of equations, execute the heuristic, and refine the mesh. For *ASMR++*, the timing also accounts for generating the observation graph and executing the trained policy. (**Left**) *ASMR++* is slower for the same number of mesh elements due to additional computational steps. (**Right**) For a given target error, *ASMR++* achieves comparable or better runtime on these environments by requiring fewer mesh elements, thanks to its higher refinement quality.

The *ZZ Error Heuristic*, which relies on element gradient information, is comparatively fast. Figure 4.11 compares it to *ASMR++* for both a fixed number of elements and for achieving a target error. While *ASMR++* is slightly slower per element due to the additional cost of generating an observation graph and executing a trained policy, its superior refinement quality leads to comparable or better runtimes for achieving a given target error on `Linear Elasticity` and `Heat Diffusion`. Additionally, the *ZZ Error Heuristic* relies on initial mesh tuning (compare Figure B.3) and exhibits significant variability in solution quality across environments. While it can be faster than *ASMR++* on environments where its heuristic happens to align well, it lacks the principled, error-focused optimization objective provided by *ASMR++* and its underlying RL framework. Prior work (Yang et al., 2023a) has also demonstrated that even simpler RL-based AMR approaches outperform it on various problem classes, particularly those involving dynamic or time-varying behavior.

4.4.4. Generalization Capabilities

We experiment with the generalization capabilities of *ASMR++* to different material parameters and unseen and larger domains during inference, as described in Section 4.3.6.

Different Domains and Load Functions. Figure 4.12 visualizes final *ASMR++* meshes for `Poisson` on different 1×1 domains and load functions that are not seen during training. We find that, without modifying the training procedure, our approach generalizes well to novel domains and load functions, likely due to the strong local inductive bias of the Swarm RL objective and the MPN policy.

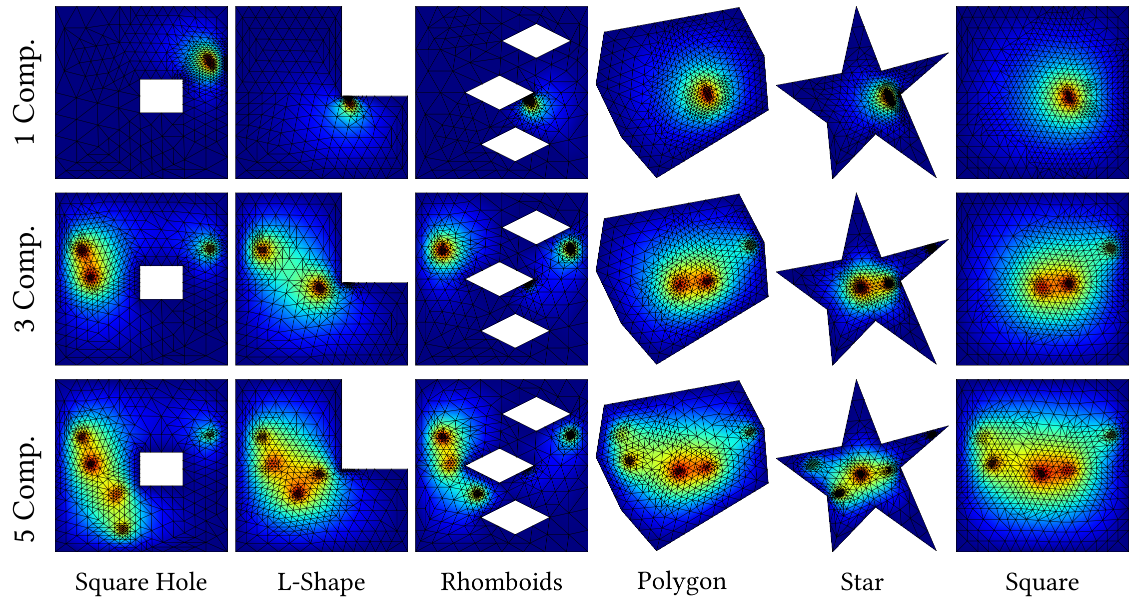


Figure 4.12.: *ASMR++* refinements for 6 different domains in $(0,1)^2$ and Poisson's equation with a GMM load function with 1, 3 and 5 components. Even though the policy is only trained on L-shaped domains with 3 components, it easily generalizes to different domains and more components in the applied load function.

Different Materials. We evaluate policies trained on each of the three variability levels (*fixed material*, *small variation*, *large variation*) explained in Section 4.3.6 on all levels, and also compare to the *ZZ Error Heuristic* on all setups. Figure 4.13 presents quantitative results. Training *ASMR++* on a limited range of parameter variations enables it to generalize effectively to a broader set of out-of-distribution parameters, while still maintaining high refinement accuracy on the fixed material parameters used in the main experiments. This generalization ability comes at no additional computational cost, since the difference between these policies is the variation in material parameters in their training data. Although *ASMR++* consistently outperforms the *ZZ Error Heuristic*, refinement quality declines slightly as parameter variability increases. This decline is likely due to the limited training data of only 100 systems of equations, and the comparatively limited capacity of the MPN backbone used for the policy. For even more complex systems of equations and generalization settings, we thus expect *ASMR++* to require a larger number of training problems and an increased latent dimensionality of the policy architecture to ensure robust and accurate refinements.

Figure 4.14 provides final refinements of *ASMR++* on exemplary systems of equations for different material parameters. For each environment, we use a single policy trained on the "large variation" environments to generate all refinements. We find that training on a range of material parameters allows *ASMR++* to provide tailored refinements on a wide range of material parameters. For example, *ASMR++* picks up on the high solution gradient in the lower left corner of the L-shape for Polypropylene, while finding that refinement is not necessary for the more evenly deforming Concrete. Additionally, since the model

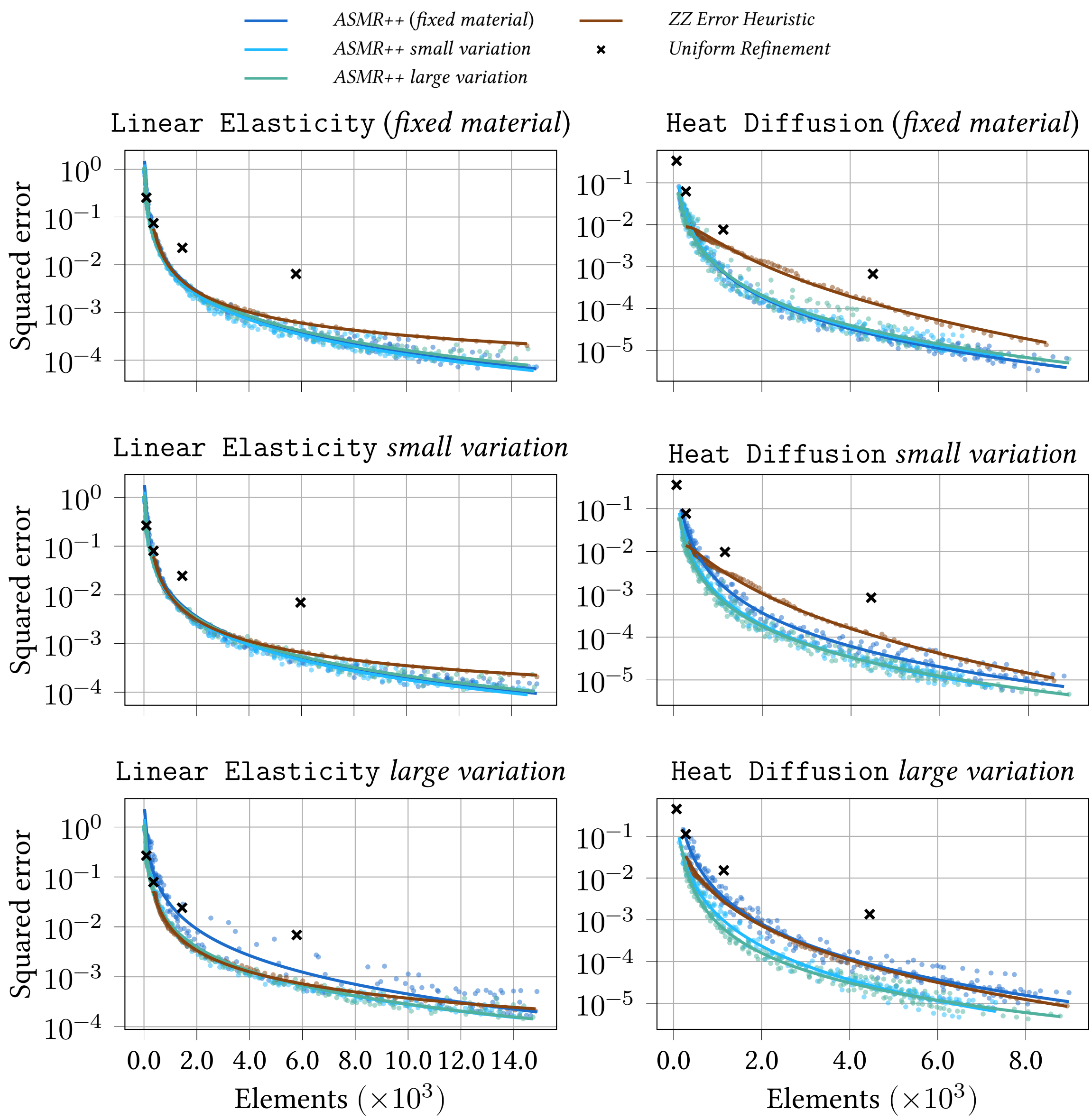


Figure 4.13.: Pareto plot of normalized squared errors and number of final mesh elements for (**left**) `Linear Elasticity` and (**right**) `Heat Diffusion`. Each sub-plot compares *ASMR++* trained on different material parameters and the *ZZ Error Heuristic* on one set of evaluation environments. More precisely, we evaluate (**top**) fixed parameters, (**middle**) small variations and (**bottom**) large variations in parameters. While *ASMR++* trained on fixed parameters struggles to generalize to unseen parameter ranges during evaluation, training on a modest range of parameters enables generalization to unseen setups while maintaining refinement accuracy on the previously seen parameter ranges.

is trained to minimize global error, it may occasionally keep secondary refinements that have a negligible impact on the total computational budget.

Larger Domains. We additionally train on a variant of `Poisson`'s equation that mimics local patches of larger domains to facilitate generalization to larger problems, as detailed in Section 4.3.6. Figure 4.15 compares *ASMR++* trained on these augmented training PDEs

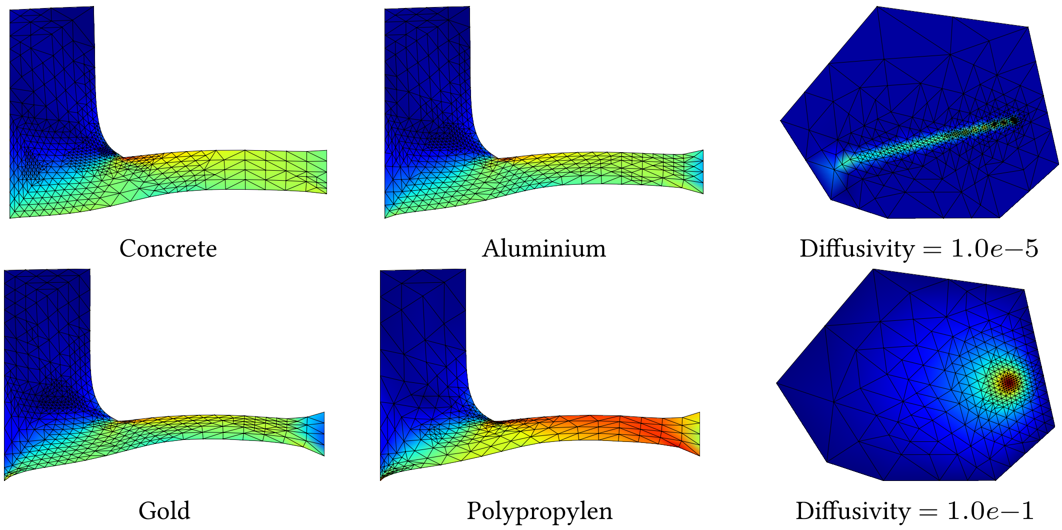


Figure 4.14.: Final *ASMR++* meshes and corresponding FEM solutions for the same `Linear Elasticity` and `Heat Diffusion` problems with varying material parameters. **Left, Middle**: Training on systems of equations with randomly sampled Poisson's ratio and Young's modulus enables the same *ASMR++* policy to deliver precise refinements for simulations on several different real-world materials, such as concrete (Young's modulus 32 GPa, Poisson ratio 0.15), gold (77.2 GPa, 0.43), aluminum (68 GPa, 0.32), and polypropylene (1.2 GPa, 0.42). The displacements in the figures have been intentionally chosen to be extreme to highlight different refinement behaviors. **Right**: Similarly, training with diverse diffusivity parameters allows for refinements for diffusivities spanning several orders of magnitude.

with the setup used throughout the paper. The results show that the augmented training setup leads to a slight decrease in performance relative to *ASMR++* trained under standard conditions. While this decrease in performance is likely caused by the more challenging optimization problem, the augmented training still yields high-quality refinements on the evaluation domains. Figure 4.17 shows a final refined mesh on a 20×20 domain with 100 components, including detailed views of specific sections. Both figures use the same policy with a fixed element penalty.

Figure 4.16 shows an exemplary *ASMR++* refinement for a 5×5 domain, while Figure 4.17 scales to a 20×20 domain. For both, the policy was trained solely on 1×1 domains. In the 20×20 domain, the refined mesh has more than 50 000 elements, which is significantly larger than any meshes seen during training, yet accurately captures the interesting parts of the solution. Here, the full refinement procedure is roughly 100 times faster than solving the fine-grained reference M^*.

Figure 4.18 shows the mesh error and inference speed for *ASMR++* policies trained on the augmented setup for `Poisson` when evaluated on domains of different sizes. For the 9×9 domain we used 10 instead of 100 evaluation PDEs due to their increased runtime and memory requirements. While the number of elements for uniform refinements is linear in the domain's volume, meshes created by our method grow less quickly to achieve the same error threshold. These results suggest that there are fewer elements with a significant error in larger domains. Accordingly, evaluating the reference mesh M^* quickly becomes expensive as the domain size gets larger, while *ASMR++* provides fast

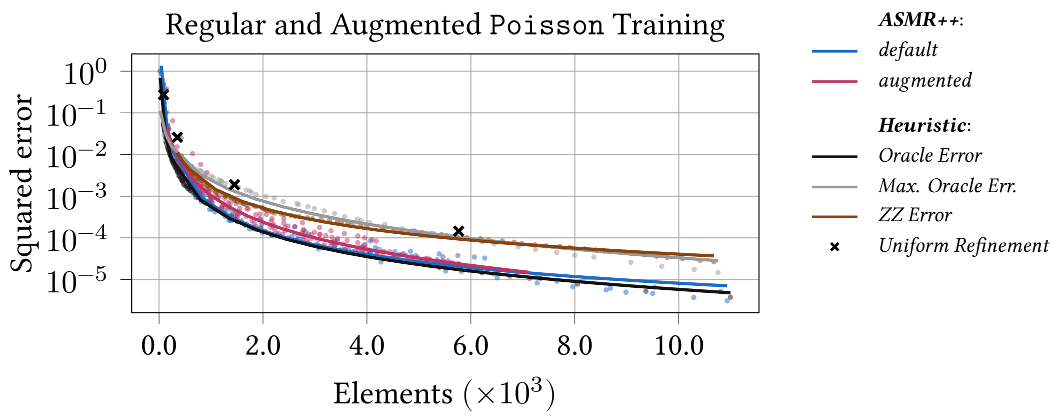


Figure 4.15.: Pareto plot of the normalized squared error for *ASMR++* trained on regular and augmented environments evaluated on the regular evaluation environments. Augmenting the training environments facilitates inference-time up-scaling at the cost of slightly decreased refinement quality.

and efficient refinements. For large meshes, our method is about 100 times faster than evaluating M^* while maintaining a mesh error of less than 0.001.

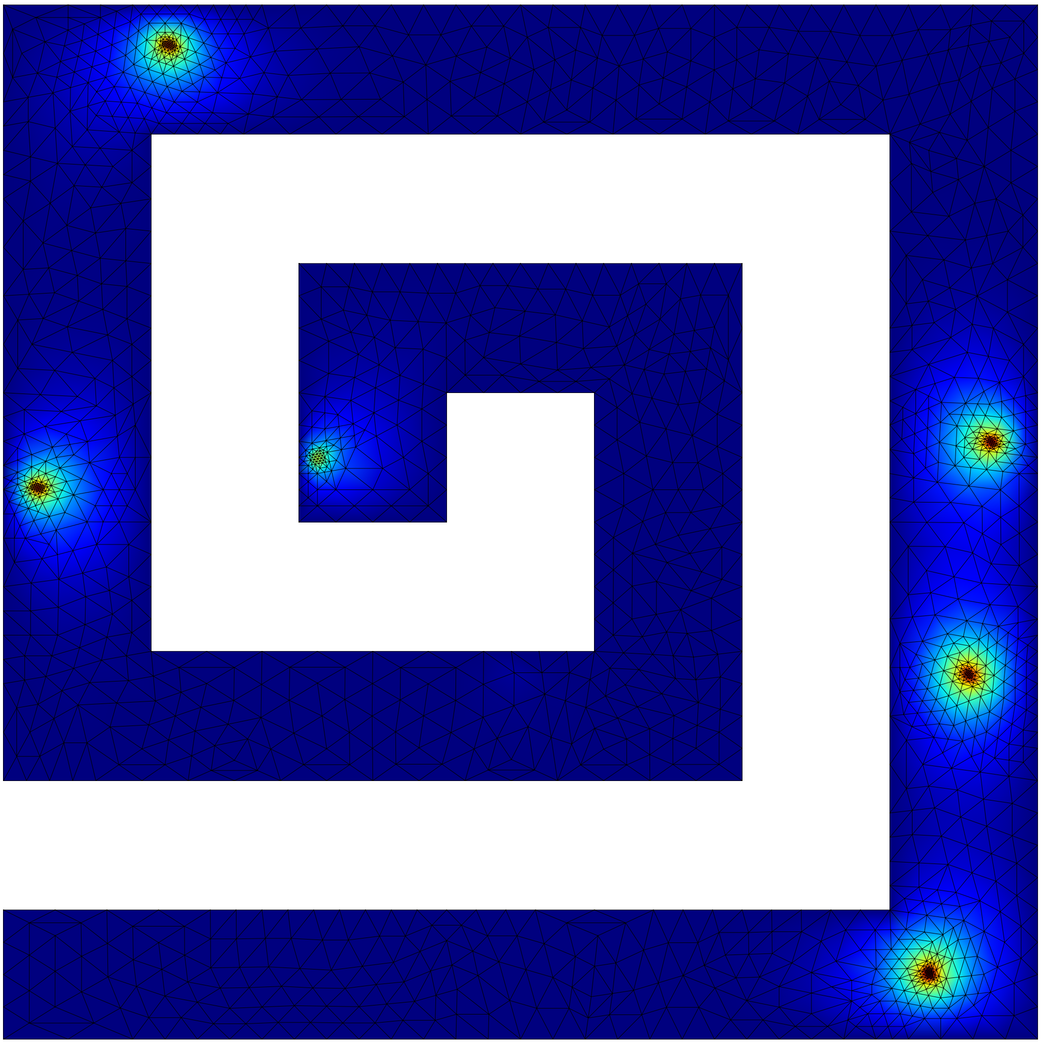
The generalization capabilities of our method are likely a result of the agent-wise optimization and the utilized MPN, which facilitate refinement strategies that are largely based on local element neighborhoods. In particular, the ability of *ASMR++* to efficiently up-scale to larger meshes during inference opens up promising applications for practical engineering applications. A policy can be trained on small and cheap environments, and then used in larger and more challenging setups during inference.

4.5. Parameter Study and Further Analysis

We conduct extensive parameter studies and several additional experiments on the challenging `Stokes Flow` environment to justify design choices and determine what makes *ASMR++* uniquely effective. Each study changes a single aspect of *ASMR++* while leaving the rest of the approach unchanged. We consider two different RL algorithms and GNN architectures each, as well as variations of the reward formulation, the agent mapping strategy and the handling of target mesh resolutions. We additionally experiment with changes in network architecture, training data diversity, and penalty scaling.

4.5.1. Proximal Policy Optimization and Deep Q-Networks

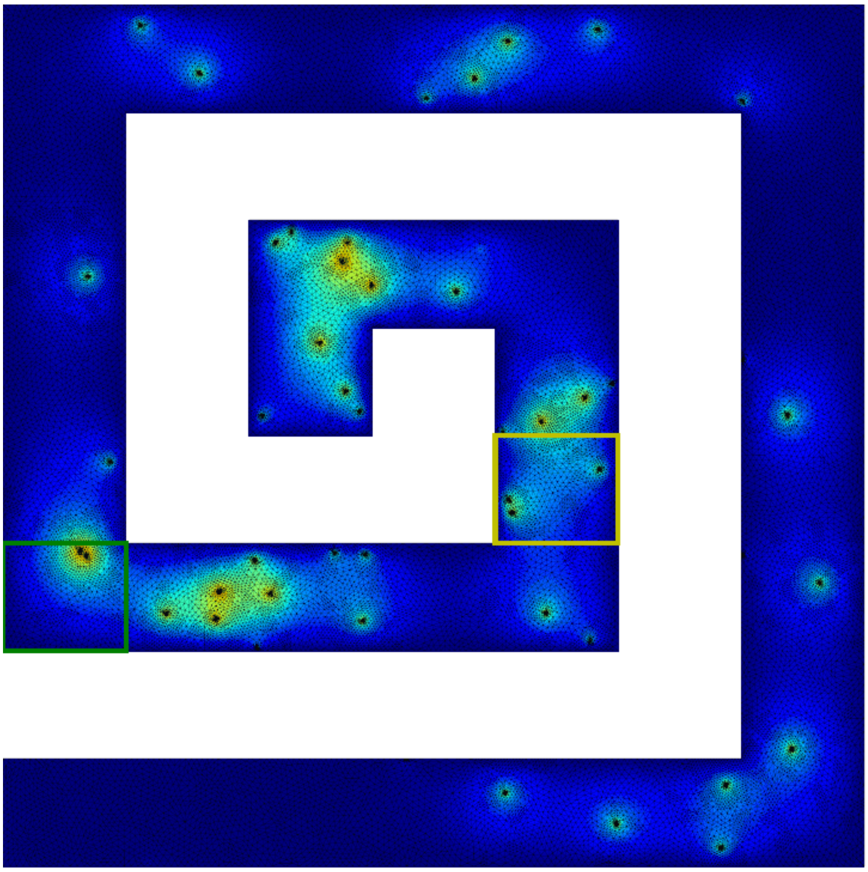
Figure 4.19 shows results on `Stokes Flow` for Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Deep Q-Network (DQN) (Mnih et al., 2013; Mnih et al., 2015) as the RL backbone for all RL-AMR algorithms. For the PPO version of *VDGN-like*, we apply the value decomposition (Sunehag et al., 2017) to the value function instead of the Q-function,



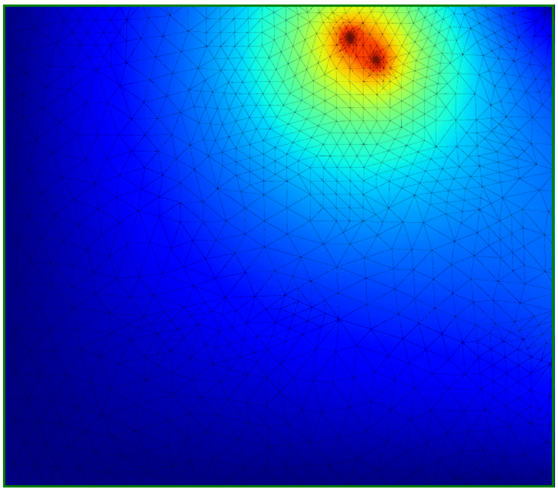
5×5 Domain

Figure 4.16.: Visualization of a final *ASMR++* mesh and corresponding solution magnitude on a 5×5 spiral domain with a randomly sampled load function. Our approach consistently provides high-quality refinements for both larger domains and more complex load functions during inference.

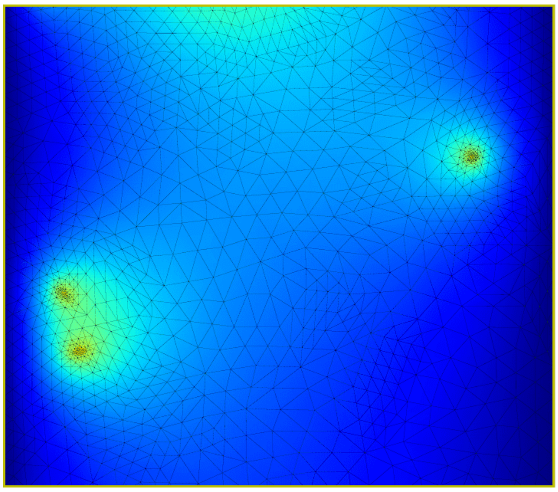
i.e., we define the value function of the full mesh as the sum of value functions of the individual mesh elements. We use a mean instead of a sum for the agent mapping of the TD error in Equation 4.3 for training the Q-value of the DQN experiments of *ASMR++*, as this experimentally increases training stability. PPO generally outperforms DQN, suggesting that on-policy optimization is favorable for the changing observation and action spaces of AMR. We thus choose PPO for all further experiments as it leads to better performance for all methods.



Full mesh



Left zoom



Right zoom

Figure 4.17.: (**Top**) Visualization of a final *ASMR++* mesh on a 20×20 spiral domain. The mesh consists of $52\,223$ elements and the full refinement and solving procedure takes about 12.2 seconds on a regular CPU. In contrast, computing the uniform mesh M^* and its PDE solution u_{M^*} takes roughly 20 minutes. (**Bottom**) Close-ups of the full mesh.

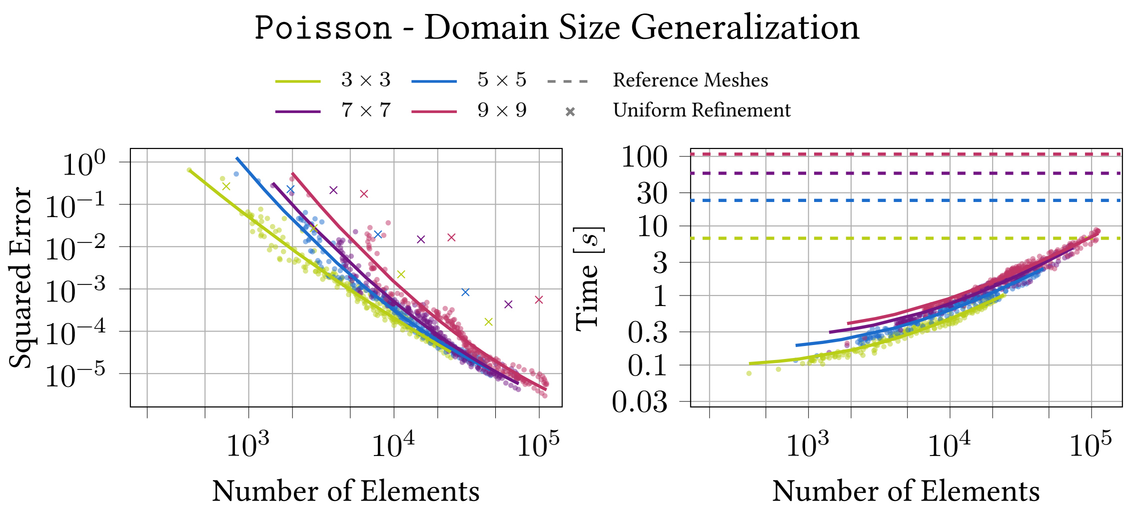


Figure 4.18.: *ASMR++* mesh error and inference speed over different sizes $N \times N$ of generalization domains after training on augmented 1×1 domains. Each evaluated domain size is denoted by a different color. (**Left**) Pareto plot of squared errors and number of final mesh elements for *ASMR++* and uniform refinements. Our method efficiently up-scales to larger domains during inference. (**Right**) Wallclock-time in seconds of *ASMR++* for different numbers of elements compared to computing the uniform reference solution u_{M^*} (dashed lines, color corresponding to domain size). While evaluating the uniform reference quickly becomes expensive, *ASMR++* provides efficient and comparatively cheap refinements even for larger domains.

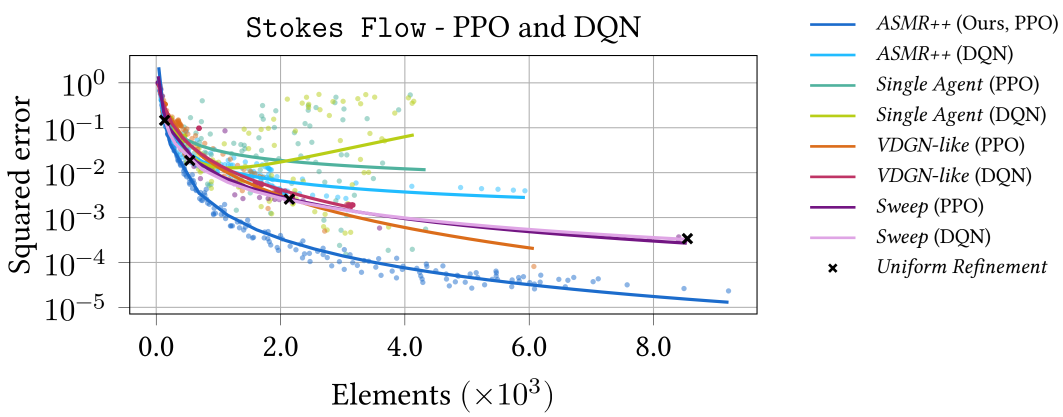


Figure 4.19.: Pareto plot of normalized squared errors and number of final mesh elements on the `Stokes Flow` for the different RL-AMR methods using PPO and Deep Q-Network (DQN) backends. The on-policy PPO outperforms the off-policy DQN for all methods. Only *ASMR++* consistently performs better than a uniform mesh.

4.5.2. Message Passing and Graph Attention Networks

Figure 4.20 compares the MPNs described in Section 2.3.3 with the modified Graph Attention Network (GAT) architecture proposed by VDGN (Yang et al., 2023b). The MPN slightly improves over the GAT for both methods, leading us to choose this architecture for all further experiments.

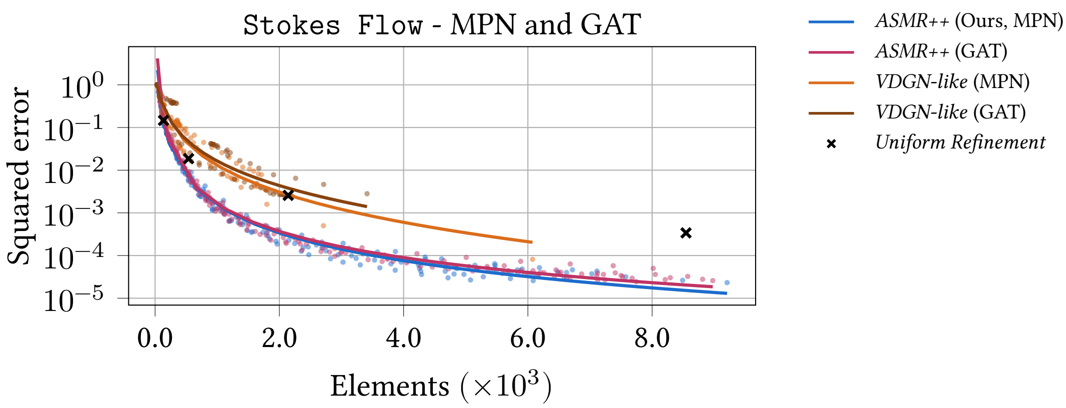


Figure 4.20.: Pareto plot of normalized squared errors and number of final mesh elements on `Stokes Flow` for *ASMR++* and *VDGN-like* for MPN and Graph Attention Network (GAT) network architectures. For both methods, the MPN shows slightly better performance, though the impact of the architecture is comparatively minor.

4.5.3. Reward Design

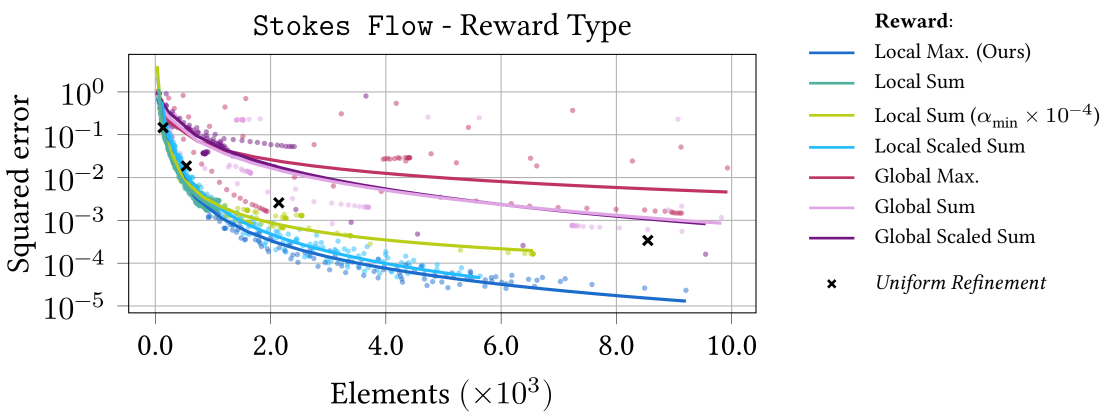


Figure 4.21.: Pareto plot of normalized squared errors and number of final mesh elements on `Stokes Flow` for different reward functions for *ASMR++*. In general, using a local reward function, i.e., individual rewards for each agent, is crucial for the performance of the method. Here, optimizing the reduction in total error per mesh element leads to sub-optimal meshes for large numbers of elements, even when significantly decreasing the minimum sampled element penalty α_{min}. Scaling this optimization with the inverse area of each mesh element as done by *ASMR* offsets this issue, but performs worse than the simpler maximum reward of *ASMR++*.

We compare the local maximum error reward of Equation 4.6 to different local and global variants. For the global variants, we obtain a scalar reward function by averaging over the local reward functions. Here, we compute the return using Equation 4.1, i.e., without mapping between agents over time. In addition to the maximum reward of Equation 4.6, we compare to the volume-scaled reward proposed by *ASMR* in Equation 4.7, and a variant that simply rewards a decrease in integrated error, i.e., Equation 4.7 without the volume scaling term.

Figure 4.21 shows the results. Using any type of global reward function leads to unstable refinements, especially for large meshes, likely because the credit for each reward cannot be properly assigned to the large number of agents in the system. For the local reward variants, the local maximum reward performs best, closely followed by the volume-scaled variant. Simply integrating the error of each mesh element and rewarding a reduction in this integrated error performs poorly for finer meshes. This result is likely the case because smaller elements generally contain fewer integration points of the reference mesh, and thus have less integrated error to reduce. Optimizing this reward produces meshes that have an even distribution of integrated element errors, rather than refinements that effectively decrease the average error on the mesh.

4.5.4. Return and Agent Mapping

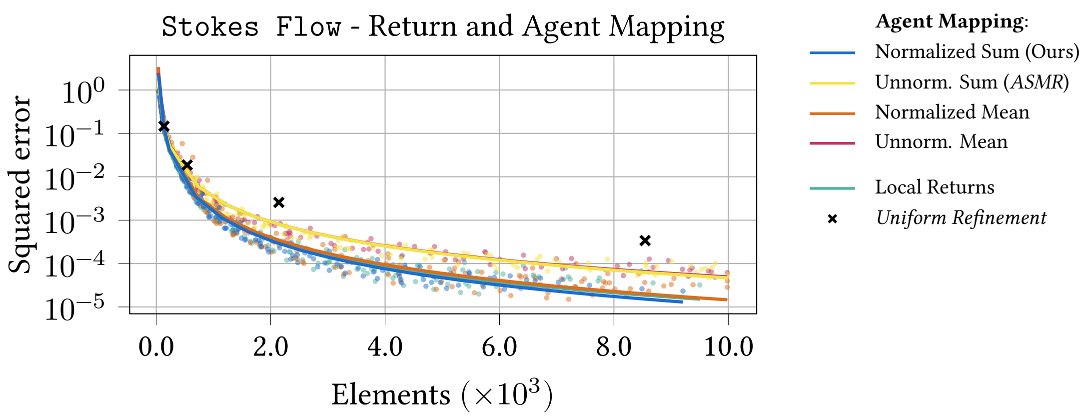


Figure 4.22.: Pareto plot of normalized squared errors and number of final mesh elements on `Stokes Flow` for different variants of the agent mapping $\mathcal{K}$ and local returns of *ASMR++*. The normalization factor in the mapping of Equation 4.4 greatly improves performance. Using a sum instead of a mean mapping additionally improves performance. Adding a global term to the returns, as done in Equation 4.8 further improves performance, indicating that a partially global objective improves global decision making of the individual agents.

Equation 4.4 enables the computation of a TD error across agents at successive time steps by mapping each agent at time step t to every agent it creates at time step $t+1$. In this setup, each spawned agent is fully attributed to its predecessor, meaning the originating agent is considered wholly responsible for any agent it generates. Compared to *ASMR*, we additionally apply a regularization to this mapping through the ratio M^t/M^{t+1}. An alternative approach would be to instead limit the total responsibility per agent to 1, achieved by averaging the mapping as

$$\mathcal{K}'^t_{ij} := \frac{\mathbb{I}(M_j^{t+1} \subseteq M_i^t)}{\sum_{j'} \mathbb{I}(M_{j'}^{t+1} \subseteq M_i^t)}.$$

We assess these two methods in Figure 4.22, finding that the normalization factor $|M^t|/|M^{t+1}|$ significantly boosts performance, likely due to its role as a regularizer during training. Furthermore, using a sum mapping, i.e., assigning each new agent a weight of 1 from its creator, proves more effective than the mean mapping, which averages the total mapping weights per old agent. We additionally explore optimizing a return that omits the global term in Equation 4.8, i.e., that only optimizes the local return of each agent. Here, we find that the partially global objective improves performance, likely because it aids in global decision making.

4.5.5. Target Mesh Resolutions

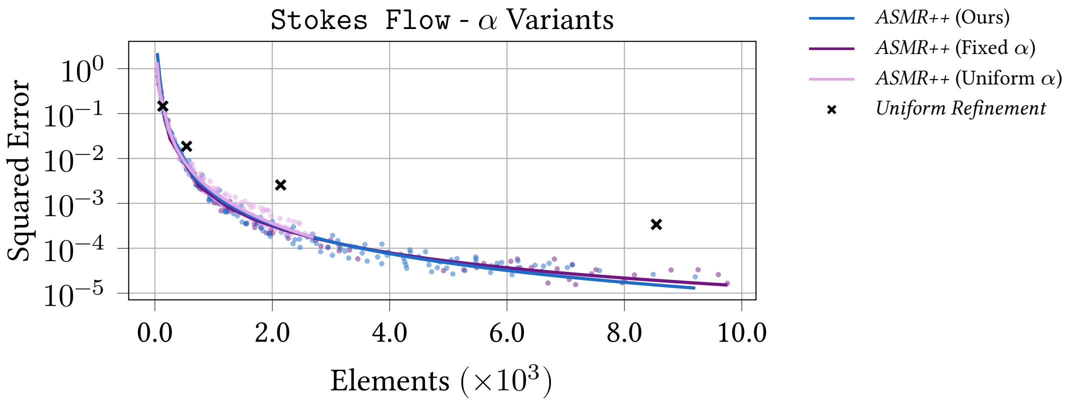


Figure 4.23.: Pareto plot of normalized squared errors and number of final mesh elements on `Stokes Flow` for different configurations of the element penalty α for *ASMR++*. Training a new policy for each value of α, as done by *ASMR* (Freymuth et al., 2023), does not significantly improve performance, which indicates the utility and benefit of training a conditional policy on a range of α values. Sampling α uniformly instead of log-uniformly during training leads to meshes with significantly fewer elements, as those values that would create large meshes are sampled less often.

Each RL-AMR method uses a single parameter to control the number of target elements of the final refined mesh. *ASMR++* and *VDGN-like* (Yang et al., 2023b) condition their policy on an element penalty α, while *ASMR* specifies a fixed element penalty as a hyperparameter. Similarly, *Sweep* uses a fixed budget N_{max} per policy, and *Single Agent* trains on a specified number of rollout steps T. Figure B.4 evaluates the impact of these parameters for *ASMR++* and the VDGN-like baseline. *ASMR++* remains stable for a wide range of penalty parameters. The penalty can be tuned by identifying the order of magnitude that corresponds to the desired number of mesh elements. This process allows users to easily control the trade-off between solution accuracy and computational cost without requiring sensitive hyperparameter tuning. Additional details are provided in Section B.4.3 and Table B.1.

Figure 4.23 compares training *ASMR++* on a fixed element penalty, as done in *ASMR* (Freymuth et al., 2023), to training on an adaptive penalty that the policy is conditioned on for each rollout. It additionally evaluates sampling the penalty uniformly instead of log-uniformly during training. Interestingly, training *ASMR++* on a fixed element penalty does not improve performance, suggesting that the *ASMR++* policy is able to learn to condition on a concrete α value. These results imply that *ASMR++* can successfully learn optimal tradeoffs between element cost and error reduction and apply it during inference. Thus, we can train a single policy for a range of mesh granularities, omitting the need for *ASMR*'s expensive re-training for each target granularity. If we train on uniformly instead of log-uniformly sampled values of α, the method predominantly provides meshes with low numbers of elements, likely because there is a logarithmic relationship between the element penalty and the number of mesh elements.

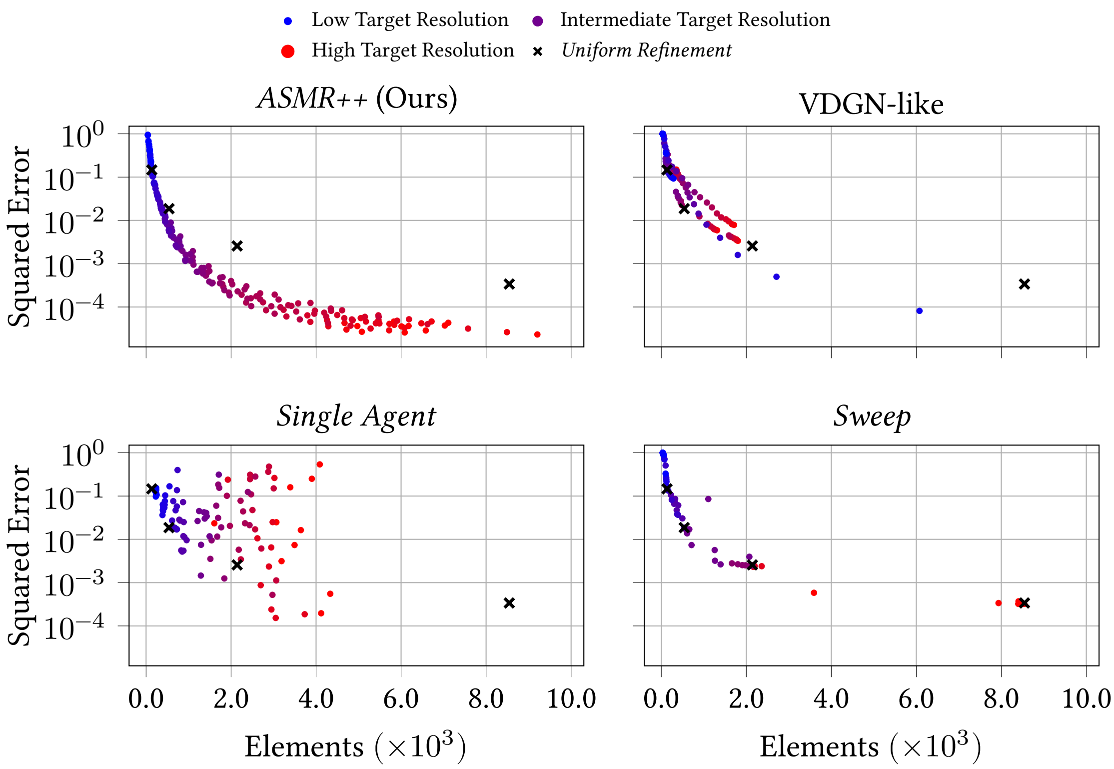


Figure 4.24.: Pareto plot of normalized squared errors and number of final mesh elements for `Stokes Flow` for all RL-AMR methods. Each plot represents a single method for different evaluations of its respective mesh resolution parameters, as detailed in Section B.4.3 and Table B.1. Small blue dots indicate meshes target with few elements, while large red dots correspond to finer target meshes. While the different RL-AMR baselines provide poor-performing or inconsistent policies, *ASMR++* yields accurate refinements and consistent numbers of final mesh elements when the policy is conditioned on any given target resolution.

Figure 4.24 visualizes the methods' performance for different target resolutions. Here, *ASMR++* provides meshes with consistent numbers of elements for a given target resolution, while the other RL-AMR baselines produce poorly refined meshes or inconsistent refinement granularities over target resolutions.

4.5.6. Further Experiments

Section B.3 studies the behavior of the *ZZ Error Error Heuristic* for a given number of initial refinments, finding that the method improves significantly for a finer initial mesh. Concretely, the method benefits from 2 uniform initial refinements. We use this variant for all experiments, and note that tuning the initial meshes is not required for any of the RL-AMR methods.

It also studies changes in the network architecture, varying numbers of training PDEs and different scales of the element penalty parameter in the reward. Here, the results indicate that as few as 10 training PDEs are sufficient to provide accurate refinements, with performance increasing for up to 100 PDEs. In terms of architecture, edge dropout (Rong et al., 2019) of 0.1, i.e., randomly omitting $10\,\%$ of edges in the mesh topology during training, slightly improves performance. Linearly increasing or decreasing the range of the element penalty α leads to meshes with fewer or more elements, respectively, yet *ASMR++* always provides stable and efficient refinements for the respective element penalty ranges.

4.6. Conclusion

We introduce Adaptive Swarm Mesh Refinement++ (*ASMR++*), an adaptive mesh refinement method leveraging Swarm Reinforcement Learning that increases the efficiency of numerical simulations with the finite element method. Our method iteratively evaluates a system of equations on a mesh, generates an observation graph from the mesh and solution, processes this graph with a message passing graph neural network, and subsequently marks mesh elements for subdivision. We treat each mesh as a homogeneous agent in a collaborative multi-agent system, training all agents with a shared policy. Crucially, we provide a per-agent reward that balances the local reduction in simulation error against the cost of adding additional mesh elements. This cost can be tuned during inference to generate meshes of different resolution using the same policy. Accommodating different mesh sizes within an episode due to subdivisions, we propose a location-based mapping that assigns each new mesh element to the element in the previous mesh that created it.

Experimental results on stationary, conforming 2D and 3D meshes demonstrate that this agent-wise approach creates high-quality refinements for meshes with thousands of elements and without requiring any expensive error estimate during inference. *ASMR++* surpasses both existing reinforcement learning-based methods and traditional refinement techniques, achieving a mesh quality comparable to expensive oracle-based error heuristics. We conduct additional experiments that highlight the benefits of coupling our per-agent reward function with our temporal agent mapping. Further, the method directly generalizes to different process conditions and domains during inference. When trained on a range of material parameters, it generalizes well to new, out-of-distribution material parameters, and when combined with simulation-specific data augmentation during training, it scales to significantly larger domains and meshes during inference. For the considered systems of equations, our method is up to 30 times faster than using a uniform refinement in small domains and can exceed a speed-up of a factor of 100 when scaling to larger evaluation setups during inference.

4.6.1. Limitations and Future Work

Our method requires solving the system of equations on the mesh following each refinement, which quickly becomes computationally expensive for finer meshes. To mitigate this, we aim to integrate auxiliary physics-based losses or policy distillation methods to implicitly learn to predict which regions to refine directly from the problem's process conditions. Additionally, our reward function is currently based on an expensive reference mesh and solution. While this reference is only required during training and *ASMR++* generalizes well from as few as 10 different training systems of equations, the maximum mesh resolution is limited by this reference. Another promising direction is to apply *ASMR++* to other numerical discretization methods such as the Finite Volume Method. Lastly, we aim to extend our method from stationary to time-dependent refinement strategies and thus include both mesh refinement and coarsening operations, as well as scale *ASMR++* to more complex non-linear problems, such as anisotropic diffusion.

5. Adaptive Mesh Generation by Iterative Mesh Resolution Prediction

While Chapter 4 introduced Adaptive Swarm Mesh Refinement (*ASMR*) as a multi-agent RL framework for local AMR, this chapter shifts the focus toward Imitation Learning (IL) and global Adaptive Mesh Generation (AMG). In *ASMR*, mesh elements act as autonomous agents guided by local rewards, which, as the previous chapter demonstrated, facilitate accurate refinements solely through exploration. However, this approach is based on local rewards, which require a concrete underlying system of equations and depend on a reference solution. The latter, in particular, incurs an exponential cost in the maximum mesh depth and thus becomes infeasible for extremely fine target meshes.

Imitation Learning (IL), as presented in Section 2.1.1, offers the use of demonstrations as a solution to both problems. By imitating the adaptive meshing decisions of an expert, we can learn at arbitrary local mesh resolutions and, if required, purely from domain geometry. This chapter proposes Adaptive Meshing By Expert Reconstruction (*AMBER*), a novel adaptive meshing method that learns to reconstruct expert meshes by iteratively constructing a set of intermediate meshes. As a direct complement to the previous chapter, *AMBER* provides a more holistic answer to our second research question, which is restated below.

Q2 How can we enable learning agents to leverage intrinsic geometric features to represent, reason over, and optimize the discretization of irregular physical domains?

Beyond the transition from RL to IL, *AMBER* is based on AMG rather than AMR. Instead of iteratively marking elements for refinement, *AMBER* thus predicts a continuous sizing field over the domain. This sizing field in turn is used to determine the target edge length of elements for a subsequently generated mesh. Since this process generates a new mesh from scratch, it allows the model to closely respect continuous features in the domain geometry. During inference, *AMBER* predicts a sequence of such sizing fields, iteratively using the induced intermediate meshes as input, until it converges to an expert-like target mesh within a few steps.

To ensure robust training, *AMBER* maintains a growing replay buffer of self-generated intermediate meshes, which are automatically labeled by projecting the expert's target sizing field onto the current mesh topology. The core of *AMBER* is a hierarchical Message Passing Network (MPN) based on the explanation in Section 2.3.3. This hierarchical architecture combines the current mesh with a coarse uniform discreitzation of the domain. This approach efficiently represents both local and global geometric features of the domain and enables *AMBER* to reason over the entire domain while maintaining high local accuracy. We use *AMBER* to accurately imitate expert meshes across a range of 2D and 3D domains, using only a few dozen demonstrations to generate meshes up to 100 000 elements. When expert demonstrations for a given system of equations are available, *AMBER* compares favorably to *ASMR*.

The following work was published as **AMBER: Adaptive Mesh Generation by Iterative Mesh Resolution Prediction** *(Niklas Freymuth, Tobias Würth, Nicolas Schreiber, Balazs Gyenes, Andreas Boltres, Johannes Mitsch, Aleksandar Taranovic, Tai Hoang, Philipp Dahlinger, Philipp Becker, Luise Kärger, Gerhard Neumann) in 38th Neural Information Processing Systems, NeurIPS 2025. Reprinted with permission of the authors. Wording, notation, structure and formulations were revised in several places.*

5.1. Introduction

Physical simulations are a fundamental tool in a wide range of science and engineering applications. As simulations become more complex, researchers and practitioners increasingly rely on numerical solutions to intricate PDEs. The FEM discretizes complex geometries into simpler mesh elements and solves the resulting system of linear equations (Brenner et al., 2008; Reddy, 2019; Larson et al., 2013; Liu et al., 2022). The FEM is ubiquitous in numerical engineering, finding application in fluid flow simulations (Connor et al., 2013), structural mechanics (Hughes, 2003; Abdullah et al., 2008), electromagnetics (Jin, 2015), and injection molding (Baum et al., 2023).

For such simulations, both the simulation cost and accuracy scale with mesh resolution. Therefore, adaptive meshing, which assigns more mesh elements to key regions of the geometry, is essential for efficient and accurate simulations (Plewa et al., 2005; Huang et al., 2010). An example is structural analysis in the automotive industry (Abdullah et al., 2008), where FEM is used to model complex components under varying forces and stresses. Figure 5.1 shows such a component, a car seat crossmember, where a finer mesh is required near bends and holes. Traditional Adaptive Mesh Refinement (AMR) techniques iteratively refine existing meshes using predefined heuristics based on problem geometry and process conditions (Zienkiewicz et al., 1992; Nemec et al., 2008; Bangerth et al., 2013). Similarly, Adaptive Mesh Generation (AMG) generates meshes from functions such as sizing fields, which define local element sizes on the geometry (Lo, 2014; Marcum et al., 2014). However, both methods are still limited in efficiency and adaptability to new applications. As a result, adaptive meshing in practice requires significant manual input and domain expertise. Engineers often hand-tune local mesh resolutions for each new geometry or problem (Lo, 2014; Shimada, 2006; Baker, 2005). This repetitive and time-consuming process creates bottlenecks in applications like iterative design and process optimization.

To address this issue, we propose Adaptive Meshing By Expert Reconstruction (*AMBER*), a data-driven method for iterative AMG. *AMBER* employs a Message Passing Network (MPN) (Gilmer et al., 2017; Pfaff et al., 2021), a class of GNNs (Bronstein et al., 2021), to predict target element sizes across a sequence of mesh refinement steps. Trained on small datasets, each consisting of roughly 20 geometries and corresponding expert meshes, *AMBER* learns underlying meshing strategies and tackles the core challenge of extreme local variation in element sizes. Unlike prior learned AMG approaches (Huang et al., 2021; Zhang et al., 2021; Khan et al., 2024), *AMBER* iteratively generates meshes using each intermediate mesh's vertices as sampling points to predict the next target sizing field. This iterative scheme, together with the MPN, enables adaptation to non-uniform geometries, while simultaneously adjusting local sampling resolution in response to previous mesh generation steps. As a result, *AMBER* is highly effective in adaptive meshing, where spatially varying target sizes necessitate correspondingly localized prediction densities.

Figure 5.1 shows an overview of our method. At inference time, *AMBER* starts from a coarse initial mesh and iteratively predicts sizing fields to feed into an out-of-the-box mesh generator (Geuzaine et al., 2009), which generates an adapted mesh. During training, predicted sizing fields are supervised by projecting element sizes from expert meshes onto intermediate meshes. To address the distribution shift introduced by intermediate meshes during inference, we maintain a replay buffer populated with meshes generated by the model itself. This strategy echoes online imitation learning approaches such as DAgger (Ross et al., 2011), which we briefly introduce in Section 2.1.1, but replaces the human-in-the-loop with automatic data generation and labeling. In doing so, *AMBER* bootstraps (Davison et al., 1997) its own training distribution, implicitly performing data augmentation (Shorten et al., 2019) by including meshes on different local scales, to stabilize learning and inference.

We evaluate our method on six novel datasets introduced in this work, covering a wide range of 2D and 3D geometries meshed by human experts and heuristics. These geometries vary in difficulty and model a diverse set of common engineering problems. We compare *AMBER* against supervised learning (Huang et al., 2021; Zhang et al., 2020b; Lock et al., 2023b) and Reinforcement Learning (RL) (Freymuth et al., 2023) baselines. *AMBER* consistently produces higher-quality meshes than all baselines, both in terms of visual quality and quantitative metrics. We additionally explore the runtime of *AMBER*'s components. We find that *AMBER*'s cost is dominated by the final mesh generation step, which is required for all mesh generation methods, and that the full *AMBER* mesh generation process is faster and scales significantly better than classical iterative error estimation methods. Furthermore, we conduct extensive parameter studies to show the effects of individual design choices, such as loss, refinement steps, and sizing field parametrization.

To summarize our contributions, we **(1)** propose *AMBER*, a novel approach for Adaptive Mesh Generation (AMG) that produces a sequence of meshes, using each intermediate mesh to predict a target resolution for the next mesh, **(2)** introduce six new datasets spanning both 2D and 3D geometries, designed to reflect realistic and diverse problem settings;

Code and datasets are available at `https://github.com/NiklasFreymuth/AMBER`

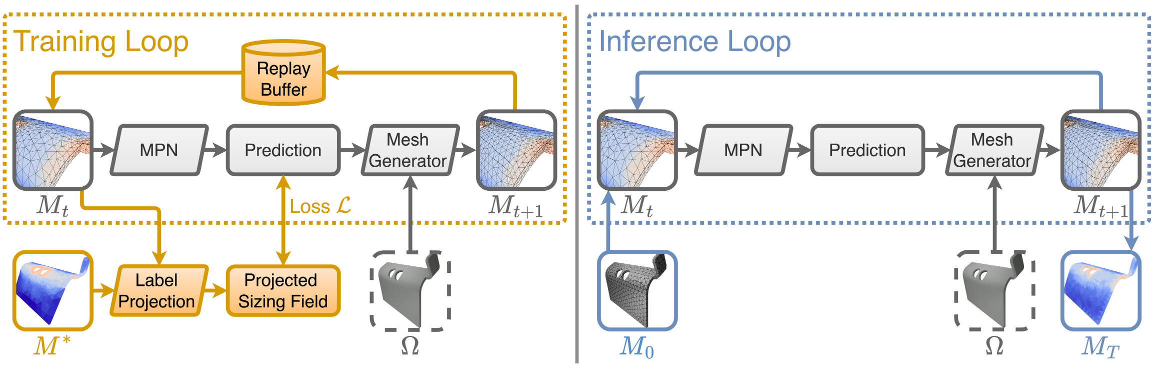


Figure 5.1.: *AMBER* learns adaptive mesh generation on complex geometries for simulation applications from an expert dataset. (**Left**) During training, *AMBER* predicts a sizing field, as indicated by the mesh's color, from labels projected from an expert mesh M^*. *AMBER* continuously updates a replay buffer with newly generated meshes to preserve a diverse and accurate training data distribution. (**Right**) During inference, *AMBER* starts from an initial mesh M_0, predicts a sizing field per element, and feeds it into a mesh generator that refines the mesh using the underlying domain Ω. This process is repeated until a final mesh M_T is produced. On the car seat crossmember shown above, *AMBER* learns that the expert assigns more mesh elements to holes and sharp bends, which are particularly interesting for strength and durability analyses.

two of these include human-generated meshes, and **(3)** conduct extensive experiments demonstrating that *AMBER* produces significantly better meshes than state-of-the-art supervised and RL methods on these datasets.

5.2. Related Work

5.2.1. Meshing for Simulation

Modern meshing approaches either use Adaptive Mesh Refinement (AMR), which refines an existing mesh (Plewa et al., 2005; Fidkowski et al., 2011), or Adaptive Mesh Generation (AMG), which generates a new mesh (Frey et al., 2007; Yano et al., 2012; Remacle et al., 2013; Si, 2008). Typical AMR techniques rely on heuristics (Zienkiewicz et al., 1992) or error estimates (Nemec et al., 2008; Bangerth et al., 2013), which can be inaccurate, unreliable, or computationally expensive (Bangerth et al., 2013; Cerveny et al., 2019; Wallwork, 2021). In contrast, AMG methods generate new meshes from geometric or solution-derived features over the domain, such as curvature or Hessian information, to prescribe local element size and potentially anisotropy (Borouchaki et al., 1998; D'Azevedo et al., 1991; Marcum et al., 2014). While effective in practice (Frey et al., 2007; Huang et al., 2010), they share the shortcomings of AMR and also require task-specific metrics or a tediously hand-crafted target sizing field for each domain (Loseille et al., 2011; Huang et al., 2010). In contrast, we aim to learn scalar sizing fields directly from expert meshes.

5.2.2. Learning-Based Mesh Generation

Existing learning-based AMG approaches train surrogate models to either directly predict a sizing field or the local solution error, which is inverted to obtain a sizing field. One line of work encodes the domain using a simple, parameterized representation, which is fed to an MLP that either predicts coordinate-conditioned outputs (Zhang et al., 2020b; Zhang et al., 2021), similar to NeRFs (Mildenhall et al., 2020), or computes the sizing field on a fixed background mesh (Lock et al., 2023b; Sanchez-Gamero et al., 2024) or as a set of point sources (Lock et al., 2023a). Huang et al. (2021) discretize the domain into a fixed-resolution image and process it with a CNN to directly predict a sizing field. More recent methods use a Graph Convolutional Network (GCN) to operate on the vertices of a coarse mesh. Of these, *GraphMesh* (Khan et al., 2024) generalizes to arbitrary polygonal domains and improves over prior GCN-based models (Kim et al., 2023). *AMBER* also predicts a sizing field on a discrete mesh, but does so iteratively across a sequence of intermediate meshes. This enables dynamic adaptation of the sizing field across scales, without being restricted to any specific domain representation or discretization, allowing it to produce higher quality meshes.

5.2.3. Learning-Based Mesh Refinement

Several recent AMR approaches employ learning for mesh refinement by subdivision, i.e., they train a model to iteratively decide which mesh elements to divide into multiple smaller elements. In this class, supervised methods include learning refinement strategies with recurrent networks (Bohn et al., 2021), optimizing element anisotropy based on error estimates (Fidkowski et al., 2021), and using hand-crafted features to estimate error for adjoint-based refinement (Roth et al., 2022; Wallwork et al., 2022). Alternatively, a recent line of work applies RL to AMR by element subdivision (Freymuth et al., 2023; Freymuth et al., 2024; Foucart et al., 2023; Yang et al., 2023a), employing carefully crafted reward functions to quantify the benefit of each refinement. These reward functions typically require an underlying system of equations and either restrict the maximum mesh resolution (Yang et al., 2023a; Freymuth et al., 2023) or encode a specific, heuristic refinement criterion (Foucart et al., 2023). Out of these methods, Adaptive Swarm Mesh Refinement (*ASMR*) (Freymuth et al., 2023; Freymuth et al., 2024) proposes local, element-wise rewards, improving scaling capability and mesh quality over previous work. *AMBER* further improves over *ASMR's* scalability and mesh quality, while avoiding the complicated reward design and the requirement for a Finite Element Method (FEM) in the loop by using expert meshes.

Another class of learning-based AMR methods employs mesh movement (Huang et al., 2010) for refinement (Song et al., 2022; Hu et al., 2024; Zhang et al., 2024). These methods start with a uniform mesh and deform its elements, requiring a fixed starting resolution. In contrast, *AMBER* learns to produce a sequence of sizing fields from a coarse uniform mesh, inducing meshes with different numbers of elements. Other mesh-movement-based methods focus on highly specific tasks, such as fluid dynamics (Yu et al., 2024; Jian et al., 2025), while *AMBER* is task-agnostic.

5.2.4. Graph Network Simulators

GNNs (Bronstein et al., 2021), particularly MPNs (Gilmer et al., 2017; Pfaff et al., 2021), are widely popular for mesh-based surrogate simulation (Pfaff et al., 2021; Linkerhägner et al., 2023; Allen et al., 2022; Allen et al., 2023; Lopez-Guevara et al., 2024; Hoang et al., 2025). MPNs encompass the function class of several classical PDE solvers (Brandstetter et al., 2022), making them a popular choice for learning representations on meshes (Pfaff et al., 2021; Linkerhägner et al., 2023; Würth et al., 2024). We similarly use MPNs on meshes, but do not learn a simulator. Instead, we generate application-specific adaptive meshes for efficient and robust FEM-based simulation.

5.2.5. Online Data Generation

Imitation learning approaches such as DAgger (Ross et al., 2011) address distribution shift by iteratively querying expert feedback on model rollouts. Bootstrapping methods like pseudo-labeling (Lee et al., 2013) and Noisy Student (Xie et al., 2020) expand the training set using model-generated labels. Replay buffers (Lin, 1992; Mnih et al., 2015; Fedus et al., 2020) mitigate covariate shift by combining past and current experiences, while data augmentation (Shorten et al., 2019; Zhang et al., 2018; Tobin et al., 2017) introduces synthetic variations to enhance generalization. Unlike these approaches, *AMBER* stores model-generated meshes across resolutions in a replay buffer and automatically projects labels onto them. This process effectively augments training data by providing meshes of different resolutions, improving distributional robustness without requiring external supervision or expert relabeling.

5.3. Method

Our training datasets contain N tuples $\{(\Omega, \mathcal{P}, M^*)\}$, each consisting of a geometry $\Omega \subseteq \mathbb{R}^d$ of dimension d, an optional set of process conditions $\mathcal{P}$, and a corresponding expert mesh M^*. Each geometry describes a closed physical body in 2D or 3D, which is discretized into simplicial elements M^*_i on the subdomain $\Omega^*_i \subset \Omega$ by the mesh. We aim to learn a function that takes a geometry Ω and process conditions $\mathcal{P}$ from the dataset and generates a mesh M that minimizes a distance metric $d(M, M^*)$ to the corresponding expert mesh M^*. We make no further assumptions on the structure of the meshes, and use both heuristically refined and human-generated meshes as expert data.

We factorize mesh generation into two parts. First, a learnable function consumes a geometry Ω, process conditions $\mathcal{P}$ and derived features, and outputs a spatially-varying, scalar-valued sizing field $\Omega \to \mathbb{R}_{>0}$. Second, a non-parametric function

$$g_{\texttt{msh}} : (\Omega \times (\Omega \to \mathbb{R}_{>0})) \to M$$

consumes a geometry and a sizing field and returns a mesh approximately conforming to this sizing field. The sizing field describes the desired average edge length of the generated mesh's elements over the domain. We consider isotropic meshes, i.e., meshes where the elements have a roughly equal aspect ratio. In this case, the local sizing field is directly related to the desired volume of the resulting mesh elements.

We refer to the non-parametric function $g_{\texttt{msh}}$ as the *mesh generator*. It creates a mesh that matches the desired sizing field under several criteria on the elements, such as their aspect ratio and size gradation. This results in well-behaved elements and a smooth transition between element sizes. While different mesh generators exist, we use the Frontal Delaunay algorithm implemented in GMSH (Geuzaine et al., 2009) for simplicity.

5.3.1. Iterative Mesh Generation with *AMBER*

Predicting a Sizing Field. Given a geometry Ω and task-specific process conditions $\mathcal{P}$, a coarse, uniform initial mesh M^t with $t = 0$ is generated. This mesh is represented as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ over mesh vertices, initialized with node and edge features $(\mathbf{h}_\mathcal{V}, \mathbf{h}_\mathcal{E})$ that encode the current mesh geometry and process conditions $\mathcal{P}$. The attributed graph is processed using an MPN as described in Section 2.3, yielding a vertex-level representation $x_j = \text{MPN}(\mathcal{G}, \mathbf{h}_\mathcal{V}, \mathbf{h}_\mathcal{E})_j = \text{MPN}(v_j)$. We then construct the *discrete* sizing field $\hat{f}(v_j)$ from x_j through a subsequent transformation.

We could alternatively predict sizing values per mesh element, yielding a piecewise constant sizing field. However, since the MPN operates on an intermediate mesh with a different topology from the target mesh, element-level predictions lack the granularity needed for effective refinement. Instead, *AMBER* predicts sizing field values over mesh vertices and applies the interpolant $\mathcal{I}_M(\hat{f})$ to yield a sizing field that is piecewise linear. This interpolant weights the discrete sizing field at the vertices v_j by the mesh's nodal basis functions $\boldsymbol{\phi}_j$ (Larson et al., 2013), yielding a *continuous* sizing field. Given a point $\mathbf{z} \in \mathbb{R}^d$ we define the interpolant as

$$\mathcal{I}_M(\hat{f})(\mathbf{z}) = \begin{cases} \sum_{j=1}^{|V|} \hat{f}(v_j)\, \boldsymbol{\phi}_j(\mathbf{z}), & \text{if } \mathbf{z} \in \Omega_i, M_i \in N(v_j) \text{ for some } j, \\ \hat{f}(v_{j'}), & \text{otherwise, where } j' = \arg\min_j \|\mathbf{z} - \mathbf{p}(v_j)\|, \end{cases} \tag{5.1}$$

where $\mathbf{p}(v_j) \in \mathbb{R}^d$ and $N(v_j) \subset M$ are the position and element neighborhood of vertex v_j, respectively. The fallback to nearest-neighbor extrapolation ensures that the sizing field is defined across all of Ω, including regions outside the discretized mesh domain.

Iterative Generation. At step t, the mesh generator consumes the continuous sizing field given by $\mathcal{I}_{M^t}(\hat{f})$ and its underlying geometry Ω. Using the vertices of each mesh as the sampling points for the next continuous sizing field and repeating this process over T steps results in a final mesh M^T. Intuitively, an accurately predicted intermediate sizing field results in a mesh that is more similar to the expert mesh, and therefore provides better sampling points for the MPN to predict the next sizing field even more accurately. Compared to one-step approaches that predict a sizing field on an image (Huang et al., 2021) or a single coarse mesh (Khan et al., 2024), *AMBER* therefore automatically adapts its sampling resolution, allowing it to output arbitrarily complex and highly non-uniform meshes where required. We prove convergence of this process in the one-dimensional case under the assumption of perfect predictions in Section C.1. The right part of Figure 5.1 visualizes this process.

5.3.2. Training *AMBER*

Predictions and Targets. Let $\text{Vol}(M_i)$ be the volume of the d-dimensional simplicial element M_i of the target mesh. We define the element-wise sizing field as the average edge length

$$f_e(M_i) = \left(\text{Vol}(M_i)\frac{d!}{\sqrt{d+1}}\right)^{\frac{1}{d}}$$

of that element. The union over the element's sizing fields induces a piecewise-constant sizing field. To compute the target value y_j of the discrete sizing field at vertex v_j of an intermediate mesh M^t, we evaluate the sizing field of the expert mesh M^* at the vertex position $\mathbf{p}(v_j)$. That is, we assign targets $y_j = f_e(M_i^*)$ where $M_i^* \in M^*$ and $\mathbf{p}(v_j) \in \Omega_i^*$. If a vertex lies outside the expert mesh due to, e.g., discretization of the domain, we project it to the nearest element. We could alternatively obtain target values by interpolating the expert sizing field using Equation 5.1. However, as we show in our experiments, due to *AMBER*'s iterative process, the local resolution of the expert mesh is sufficient to adequately represent the granularity of the solution everywhere.

We train a single shared MPN to regress the target sizing field of the current mesh generation step using a simple Mean Squared Error (MSE) loss. Since sizing fields are strictly positive, we add a softplus transformation to the network's output. To increase the weight of numerically smaller elements in the loss function, we optimize in the untransformed space. Thus, given a prediction $x_j = \text{MPN}(v_j)$, our loss becomes

$$\mathcal{L} = \frac{1}{|V|}\sum_{j=1}^{|V|}\left(x_j - \text{softplus}^{-1}(y_j)\right)^2. \tag{5.2}$$

We then recover the discrete predicted sizing field as $\hat{f}(v_j) = \text{softplus}(x_j)$.

Replay Buffer. During inference, *AMBER* auto-regressively produces a series of intermediate meshes M^t. The initial mesh M^0 is coarse and uniform. However, the corresponding

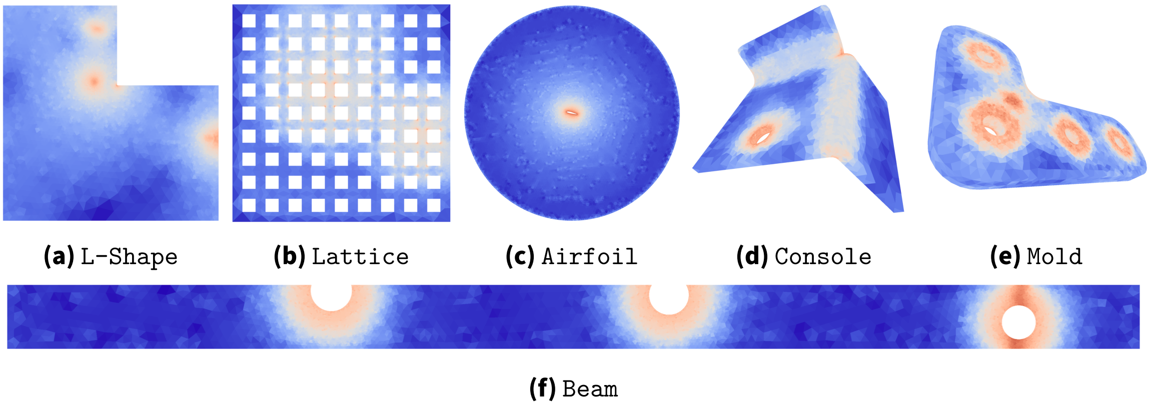


Figure 5.2.: Exemplary *AMBER* meshes for each dataset. The color represents the local element size, with smaller elements being red. We propose six novel and challenging datasets for mesh generation. **(a)** `L-Shape` uses an L-shaped domain with a multimodal load function. **(b)** `Lattice` features parameterized 2D lattices with complex Dirichlet boundaries. **(c)** `Airfoil` includes geometries representative of aerodynamic flow setups. **(d)** `Console` consists of 3D car seat crossmembers. **(e)** `Mold` includes complex 3D plates used in injection molding contexts. **(f)**`Beam` covers elongated, perforated beams inducing long-range mesh dependencies.

expert mesh M^* is generally finer and has highly varied topology. To prevent a distribution shift between the training data and the data seen during inference, we therefore maintain a replay buffer (Lin, 1992; Fedus et al., 2020) of bootstrapped data containing intermediate meshes that *AMBER* generates during training. The replay buffer is initialized with one uniform coarse mesh per expert mesh. After each training epoch, we sample k meshes from the replay buffer for producing new intermediate meshes. For each, we predict a discrete sizing field, generate a new mesh from the induced continuous sizing field, annotate the vertices with a target sizing field using the expert mesh, and store this new labeled mesh in the buffer. This iterative data generation is similar to online imitation learning approaches such as DAgger (Ross et al., 2011), as introduced in Section 2.1.1, but uses automatic data generation and labeling. The full training pipeline is shown on the left of Figure 5.1.

5.3.3. Empirical Improvements

Inspired by common best practices, we propose several algorithmic optimizations to further improve *AMBER*'s applicability and efficiency.

Uniform Refinement Depth. During training, we assign each intermediate mesh a *depth* that corresponds to the number of refinement steps it has undergone from the initial uniform mesh. To reduce distribution shift between inference and training, we enforce a uniform distribution over mesh depths in the replay buffer. When generating new intermediate meshes, we first uniformly sample a target depth and then a mesh with the corresponding depth. We set the maximum depth to the number of refinement steps used during evaluation, T.

Adaptive Batch Size. Since the meshes in the replay buffer vary greatly in size, using a fixed number of meshes per batch would sometimes lead to out-of-memory errors, or otherwise leave significant available memory unused. Instead, we set a maximum total size over all graphs in a batch, and greedily fill a batch with the least-sampled meshes until the limit is reached. We define the size of a graph as the sum of its number of nodes and edges $s(\mathcal{G}) = |\mathcal{V}| + |\mathcal{E}|$.

Hierarchical Architecture. The receptive field of an MPN is determined by the number of message passing steps. As a mesh undergoes iterative refinement, the receptive field can vary significantly across the domain. This makes it challenging to choose appropriate hyperparameters and hinders long-range communication between regions of the graph during the later refinement steps. To ensure a consistent, resolution-invariant receptive field, we employ a hierarchical graph structure that combines the graph $\mathcal{G}^0 = (\mathcal{V}^0, \mathcal{E}^0)$ corresponding to the initial coarse mesh M^0 with that of the current intermediate mesh M^t for all $t > 0$. The hierarchical graph is defined as

$$\mathcal{G}_{\text{hier}} = \left(\mathcal{V}^0 \cup \mathcal{V}^t,\ \mathcal{E}^0 \cup \mathcal{E}^t \cup \mathcal{E}^{\text{cross}}\right), \quad \text{with } \mathcal{E}^{\text{cross}} = \{(v, \rho(v)),\ (\rho(v), v) \mid v \in \mathcal{V}^t\}.$$

The cross-level edges $\mathcal{E}^{\text{cross}}$ connect each intermediate vertex $v \in \mathcal{V}^t$ and its closest vertex $\rho(v) = \arg\min_{u \in \mathcal{V}^0} \|\mathbf{p}(v) - \mathbf{p}(u)\|_2$ in the coarse mesh. We provide a binary node feature indicating the current mesh M^t, and mask all node-level features of the initial mesh, using it solely to provide consistent topological connectivity.

Input/Output Normalization. We normalize all network inputs, i.e., all node and edge features, to have zero mean and unit variance. The labels are normalized similarly, and the inverse normalization is applied to map predictions back to the original scale. Since the data distribution evolves as new meshes are added to the replay buffer, we maintain running statistics for each input and target feature.

Residual Prediction. We improve training stability by predicting the residual between the target sizing field $y_j = f_e(M_i^*)$ and the current discrete sizing field $b_j = f(v_j)$. Given the element neighborhood $N(v_j)$ of vertex v_j and element-based sizing fields $f_e(M_i)$, we compute the current discrete sizing field b_j at v_j from the current mesh as the convex combination

$$b_j = f(v_j) = \sum_{M_i \in N(v_j)} \frac{\text{Vol}(M_i)}{\sum_{M_i \in N(v_j)} \text{Vol}(M_i)} f_e(M_i). \tag{5.3}$$

We now recover the predicted sizing fields by $\hat{f}(v_j) = \text{softplus}(x_j + \text{softplus}^{-1}(b_j))$, and adapt the loss in Equation 5.2 accordingly.

Scaling Sizing Fields. We can scale the resolution of generated meshes by introducing a simple refinement constant c_t depending on the generation step t, such that the next generated mesh is $M^{t+1} = g_{\texttt{msh}}(\Omega, c_t \mathcal{I}_{M^t}(\hat{f})(\mathbf{z}))$. While the predicted intermediate meshes allow *AMBER* to adaptively refine its sampling resolution, reaching the full resolution of the expert mesh is unnecessary and computationally expensive. To mitigate this, we set $c_t{>}1$ for $t{<}T{-}1$ to coarsen intermediate meshes, starting at the first step $t{=}0$. Here, setting an exponentially decaying c_t reduces the number of elements for intermediate meshes without reducing the accuracy of the final mesh M^T. Additionally, during inference, we can also set $c_{T-1}<1$ to generate meshes that have a higher resolution than the expert. This adaptation allows the model to flexibly adapt to a given element budget without retraining, which enables zero-shot generalization via a single scalar parameter.

5.4. Experiments

5.4.1. Datasets

We introduce six novel datasets representing realistic FEM problems that need adaptive meshing to meet common efficiency and accuracy requirements. The datasets span 2D and 3D domains, as well as diverse applications in physics-based simulation, structural mechanics, and industrial design. Depending on the dataset, we generate geometries procedurally, or source them from openly available or custom datasets. For datasets without a concrete underlying system of equations, we generate meshes from human experts and manually designed, specialized heuristics. Other datasets consider a concrete problem, where we employ an iterative refinement heuristic that utilizes a FEM error indicator. Using this heuristic, we create *easy*, *medium*, and *hard* variants of the `L-Shape` dataset to provide expert meshes on different scales. Here, more refinement steps result in an expert mesh with more elements and a larger difference between the largest and smallest elements increases, making the dataset more challenging. Figure 5.2 illustrates representative *AMBER* meshes from the test set of each dataset. Across datasets, mesh resolution ranges from 1 042 to 65 191 elements.

5.4.2. Setup

MPN Architecture. The MPN of *AMBER* consists of 20 separate message passing steps for all datasets. Each message passing step uses separate two-layer MLPs and LeakyReLU activations for its node and edge updates. We apply Layer Normalization (Ba et al., 2016) and Residual Connections (He et al., 2016) independently after each node and edge feature update, and use Edge Dropout (Rong et al., 2019) of 0.1 during training. The final node features are fed into a two-layer MLP decoder. All MLPs use a latent dimension of 64. We experimented with slightly different parameterizations in preliminary experiments, finding that *AMBER* is relatively insensitive to the details of the underlying MPN.

Training. We implement all neural networks in `PyTorch` (Paszke et al., 2019) and optimize using ADAM (Kingma et al., 2015). We use a learning rate of $1.0e{-}3$ and a linear learning rate scheduler with a warmup from 0 to the full learning rate during the first $10\,\%$ of training. We apply weight decay of $1.0e{-}6$. We train for a total of 25 600 mini-batches for `L-Shape`, `Lattice` and `Airfoil`, and 51 200 mini-batches for `Beam`, `Console` and `Mold`. Section C.3 provides a detailed overview of all training and model hyperparameters.

Input Features. Each node is assigned features for the average sizing field of adjacent elements, as provided in Equation 5.3, and the vertex degree. As edge features, we use the Euclidean distance between vertex positions and an approximate curvature, defined as the signed angle between the averaged surface normals of the edge's endpoints. The curvature lies in $[-1, 1]$, with positive values for convex and negative values for concave regions.

We further derive dataset-specific features as a function of the process conditions $\mathcal{P}$ for `L-Shape`, `Lattice` and `Mold`, as detailed in Section C.2. The `L-Shape` and `Lattice` datasets use a FEM solver in the loop for expert mesh generation via an iterative refinement heuristic. For these datasets, we therefore provide FEM solutions as a vertex-level input feature for each mesh. Since all features are invariant to Euclidean transformations, the architecture is invariant to rotation, translation, reflection, and vertex permutation (Bronstein et al., 2021).

Mesh Generation. We use GMSH (Geuzaine et al., 2009) for mesh generation. For simplicity, we clip the predicted sizing fields during mesh generation to

$$(0.8 \min\{f_e(M_i^*)\},\ 1.25 \max\{f_e(M_i^*)\}), \quad \text{with } M_i^* \subseteq M^*,\ M^* \in \mathcal{D},$$

i.e., to a range around the most extreme values seen in the training dataset $\mathcal{D}$. Here, f_e is the element-wise sizing field introduced in Section 5.3.2. This is only done during mesh generation, and does not impact the model predictions or the loss of Equation 5.2. We further constrain the mesh generation process of *AMBER* to a budget of $1.5 \max\{|M_i^*|, M_i^* \subseteq M^*, M^* \in \mathcal{D}\}$ elements, i.e., to $150\,\%$ of the mesh elements of the largest mesh in the training dataset. To ensure that this budget is met, we employ a conservative heuristic that estimates the number of elements in a newly generated mesh from a given sizing field, and then computes a scaling factor such that the new mesh does not exceed the available number of elements. This constraint makes training more predictable by preventing very large meshes and thus unexpected peaks in runtime between training epochs. While this constraint is also active during inference, we find that it practically never activates after the training has converged.

Hardware and Compute. All graph-based methods are trained on an NVIDIA 3090 GPU. The image-based methods are instead trained on an NVIDIA A100 GPU to accommodate the memory requirement of the comparatively high-resolution images. Each method is given a computational budget of up to 36 hours, although most methods, including *AMBER*, usually converge after 4-12 hours, depending on the considered dataset. We train every method for five seeds. We evaluate four methods on eight datasets, counting `L-Shape` (*easy*/*medium*/*hard*) separately, and four additional methods on three datasets. We additionally have a total of 31 additional experiments across three datasets. Combined, this yields an estimated total compute of $8[\text{hours}] \times 5[\text{seeds}] \times (8\times4+4\times3+31) = 3000[\text{hours}]$. A comparable amount was used for preliminary runs and hyperparameter tuning.

5.4.3. Evaluation Metrics

We evaluate the generated meshes by comparing their local resolution to that of an unseen expert reference mesh on the same geometry and process conditions, using five random seeds per experiment.

Density-Aware Chamfer Distance (DCD). We primarily evaluate mesh similarity using the Density-Aware Chamfer Distance (DCD) (Wu et al., 2021), a symmetric, exponentiated variant of the Chamfer distance that accounts for multiple points in one set matching a single point in the other. Given vertex sets $\mathcal{V}_1$ and $\mathcal{V}_2$, the DCD is defined as

$$d_{\text{DCD}}(\mathcal{V}_1, \mathcal{V}_2) = \frac{1}{2}\Bigg[\frac{1}{|\mathcal{V}_1|}\sum_{v\in\mathcal{V}_1}\left(1 - \frac{1}{n_v}e^{-\|\mathbf{p}(v)-\mathbf{p}(\hat{v}(v,\mathcal{V}_2))\|_2}\right) + \frac{1}{|\mathcal{V}_2|}\sum_{v\in\mathcal{V}_2}\left(1 - \frac{1}{n_v}e^{-\|\mathbf{p}(v)-\mathbf{p}(\hat{v}(v,\mathcal{V}_1))\|_2}\right)\Bigg], \tag{5.4}$$

where $\hat{v}(v, \mathcal{V}') = \arg\min_{v'\in\mathcal{V}'} \|\mathbf{p}(v) - \mathbf{p}(v')\|_2$ is the nearest neighbor, and n_v is the number of points in the other set for which v is the nearest neighbor. The DCD is a purely geometric metric that treats both vertex sets as samples from an unknown density over the domain.

L^2 **Error.** We additionally evaluate mesh similarity using a symmetric relative projected L^2 error between the vertex-based sizing fields of the evaluated and expert meshes. This metric complements the purely geometric DCD by quantifying discrepancies in local element sizes. Let f and f^* denote the vertex-based sizing fields of Equation 5.3 on the evaluated mesh M and expert mesh M^*, respectively. We use the interpolant $\mathcal{I}$ from Equation 5.1 to evaluate each sizing field at the vertex positions of the opposite mesh. The symmetric relative projected L^2 error is then defined as

$$d_{\text{L2}}(M, M^*) = \frac{1}{2}\left(\frac{\left\|f^*(v_j^*) - \mathcal{I}_{M^*}(f)(\mathbf{p}(v_j^*))\right\|_2}{\left\|\mathcal{I}_{M^*}(f)(\mathbf{p}(v_j^*))\right\|_2} + \frac{\left\|f(v_j) - \mathcal{I}_{M}(f^*)(\mathbf{p}(v_j))\right\|_2}{\left\|\mathcal{I}_{M}(f^*)(\mathbf{p}(v_j))\right\|_2}\right), \tag{5.5}$$

where $\|\cdot\|_2$ denotes the discrete ℓ^2 norm over vertices. The combination of these two metrics with different semantic interpretations is robust against potential artifacts in the generated meshes.

Error Indicator Norm. Finally, for `L-Shape`, we evaluate *ASMR* and *AMBER* using the norm of the error indicator of Equation C.7, i.e.,

$$d_{\text{err}}(M) = \|\text{err}(M_i)\|_2 = \sqrt{\sum_{M_i\in M}\text{err}(M_i)^2}. \tag{5.6}$$

In contrast to the above metrics, the error indicator norm approximates the remaining simulation error for a given mesh, independent of some reference mesh or vertex set. It naturally decreases for finer meshes, but quantifies how well a given mesh works for downstream simulation for its budget. We thus evaluate the Expert, *ASMR* and *AMBER* for different target mesh granularities on a Pareto front of number of mesh elements compared to this norm.

5.4.4. Baselines and Variants

We compare to two supervised methods, *GraphMesh* (Khan et al., 2024) and *Image* (Huang et al., 2021), as well as an Reinforcement Learning (RL) baseline, Adaptive Swarm Mesh Refinement (*ASMR*) (Freymuth et al., 2023; Freymuth et al., 2024). Unless mentioned otherwise, all baseline and variant experiments follow the setup and hyperparameters described in Section C.3.

We adapt both supervised baselines to use softplus-transformed predictions. This transformation is omitted in the original works, which focus on relatively simple problems where training instabilities are less pronounced. Without it, models tend to diverge, producing overly fine meshes. To disentangle training and algorithmic design, we additionally introduce *Variants* of each supervised baseline. These *Variants* incorporate our loss (Equation 5.2) and normalization. Detailed descriptions for each baseline are provided below.

GraphMesh. Our first baseline, *GraphMesh* (Khan et al., 2024) uses a two-stage GCN (Kipf et al., 2017) to extract geometric information from polygonal domains. It constructs a local copy of the boundary graph for each coarse mesh vertex, encoding relative features to all boundary vertices. These features are mean value coordinates (Floater, 2003), spatial distances, and mesh-hop counts. Thus, each coarse vertex is represented by an individual boundary graph that contains features of the boundary relative to this vertex. This construction limits *GraphMesh* to polygonal domains, which in our case restricts it to the `L-Shape` datasets. These graphs are processed by a single-layer GCN, and the resulting embeddings are pooled to yield one latent vector per coarse vertex of the original mesh.

To enable load-specific sizing field prediction, the same vertex-level features used in *AMBER* are appended to these embeddings. For the `L-Shape` datasets, these features include vertex degree, interpolated sizing field, load function value, and solution value at the vertex position. The combined features are used as node inputs to a second GCN stage consisting of 6 residual graph convolutional layers with 128 dimensional hidden states. *GraphMesh* is trained using a Mean Average Error to the target sizing field and does not apply normalization. We find that *GraphMesh* quickly starts to overfit, especially on `L-Shape` (*easy*), likely due to poor generalization capabilities of its GCN and the construction of the geometry embedding. To compensate, we reduce the number of training steps to 3 200/6 400/12 800 for `L-Shape` (*easy*/*medium*/*hard*). We tune the resolution of the underlying mesh by dataset for optimal validation performance.

In *GraphMesh (Variant)*, we instead apply the loss in the inverse-softplus space, as in Equation 5.2, and add input/output normalization. We also use 20 layers with dimension 64 to match *AMBER*'s MPN.

***Image* Baseline.** We also consider pixel- and voxel-wise inputs with *Image* (Huang et al., 2021). The method predicts sizing fields from binary geometry masks of a discretized domain using a 2D or 3D CNN, respectively. In 2D, we use 512 pixels along the longest axis. We evaluate other resolutions in Section C.4.1. In 3D, we use 96 voxels along the longest axis. We follow the original setup and use a U-Net (Ronneberger et al., 2015) with 64 initial channels and 5 down- and up-convolution blocks. Each block contains 2 convolutions with kernel size 3, followed by batch normalization and a ReLU activation. After each down-convolution, we apply max-pooling with kernel size and stride 2 to halve the resolution and double the number of channels. The up-convolutions reverse this process, and skip connections are added between corresponding depths. We use 2D and 3D convolutions, batch normalization and pooling operations for the 2D and 3D datasets, respectively.

For problem-specific datasets, i.e., `L-Shape` and `Lattice`, we generate a uniform background mesh with roughly one element per pixel and compute an FEM solution on this mesh to yield our input features. For `L-Shape`, we additionally include the load function evaluated at each pixel. Finally, for all datasets, we add a binary mask that indicates if a given pixel or voxel is inside the domain as an input feature. We also mask the loss accordingly, only predicting and training on pixels within the domain. The *Image* baseline is trained on a regular MSE loss over pixel-wise predicted and target sizing fields.

The *Image (Variant)* extends the *Image* baseline to the loss of Equation 5.2 and input/output normalization. We evaluate both choices individually in Section C.4.1.

AMBER (1-Step). To demonstrate the benefit of iterative mesh generation, we experiment with a variant of *AMBER* that only uses a single mesh generation step, i.e., that sets $T=1$. This variant predicts a vertex-level sizing field on a uniform mesh, and uses this prediction to directly generate the adaptive mesh. The resulting *AMBER (1-Step)* explores *AMBER* without the ability to generate and act on an intermediate meshes, limiting it to a fixed sampling resolution for the predicted sizing field. We keep all *AMBER* hyperparameters the same, but omit all parts of the method that depend on iterative mesh generation. Since *AMBER (1-Step)* heavily depends on the resolution of its input mesh, we tune this resolution separately for each dataset for optimal validation performance.

Adaptive Swarm Mesh Refinement (*ASMR*). We compare against Adaptive Swarm Mesh Refinement (*ASMR*) (Freymuth et al., 2023; Freymuth et al., 2024) as an RL baseline. *ASMR* learns a policy to iteratively mark elements for AMR, optimizing a reward function tied to the improvement of a specific FEM solution. We evaluate *ASMR* on the `L-Shape` dataset, which we implement in the provided codebase. For consistency with *AMBER*'s experimental setup, we replace the original mesh generator with GMSH and adapt the dataset parameters to match those in Section C.2.1. We further adopt *AMBER*'s batching scheme to prevent too-large batches, sampling from the RL replay buffer until the combined number of graph nodes and edges reaches 500 000.

We employ *ASMR*'s volume-scaled reward function (Freymuth et al., 2023), as defined in Equation 4.7. This reward requires a fine-grained uniform reference mesh to compare to, whose resolution bounds the maximum number of refinements. We obtain this reference mesh by uniformly refining the initial mesh six times, finding further refinements to be computationally infeasible. Section C.4.2 explores a *ASMR (Error Indicator)* variant that omits the reference mesh in favor of the error indicator of Equation 5.6. This modification

`https://github.com/niklasfreymuth/asmr` enables deeper refinement and mesh resolutions comparable to those of the expert. In both cases, we evaluate the quality of the generated mesh using the error indicator of Equation 5.6. This metric slightly differs from the integrated error used in *ASMR*'s objective due to its volume scaling. However, in preliminary experiments, we found that removing the scaling term led to unstable training and non-convergence.

The policy backbone is a 2-step MPN with hidden dimension 64. Preliminary experiments with more message passing steps showed no improvement, which is consistent with prior observations on RL model scaling (Bjorck et al., 2021; Nauman et al., 2024a). Finally, we condition the policy on an adaptive element penalty α to allow different target mesh sizes. During training, we sample α from a predefined range that yields mesh sizes comparable to `L-Shape` (*easy*/*medium*/*hard*). At inference time, we evaluate across a range of 20 geometrically spaced α values, producing meshes of varying resolution and corresponding indicator error for a full comparison.

5.4.5. Runtime and Cross-Dataset Generalization

We measure the runtime of *AMBER* and its individual components for `L-Shape` (*easy*/*hard*) and compare it to the expert heuristic used to generate the data. We additionally explore *AMBER*'s ability to generalize across datasets by training a single model on joint data of `L-Shape` (*hard*), `Lattice` and `Airfoil`. We concatenate the 20 expert meshes per dataset into 60 total training meshes, using a shared replay buffer for the data. The dataset is one-hot encoded in the node features, and dataset-specific features are zeroed out when unavailable. Other training and inference hyperparameters remain unchanged. We call this variant *AMBER* (*Mixed*).

5.5. Results

5.5.1. Quantitative Results

Density-Aware Chamfer Distance (DCD) Evaluations. Figure 5.3 evaluates Density-Aware Chamfer Distance (DCD) over vertex sets to the expert mesh across datasets. On `L-Shape` (*easy*), all methods perform well. As complexity increases for, e.g., `L-Shape` (*medium*/*hard*), our training procedure shows more significant benefits, causing both variants to significantly outperform their published baselines. Across datasets, the *AMBER (1-Step)* produces accurate sizing field predictions and high-quality meshes closely matching the expert. It also generalizes to 3D, where the *Image* methods struggle. *AMBER* further improves mesh quality, likely due to its iterative mesh generation. Here, multiple generation steps allow the intermediate meshes, which govern the prediction resolution, to adapt dynamically to the underlying geometry, improving mesh quality in complex regions.

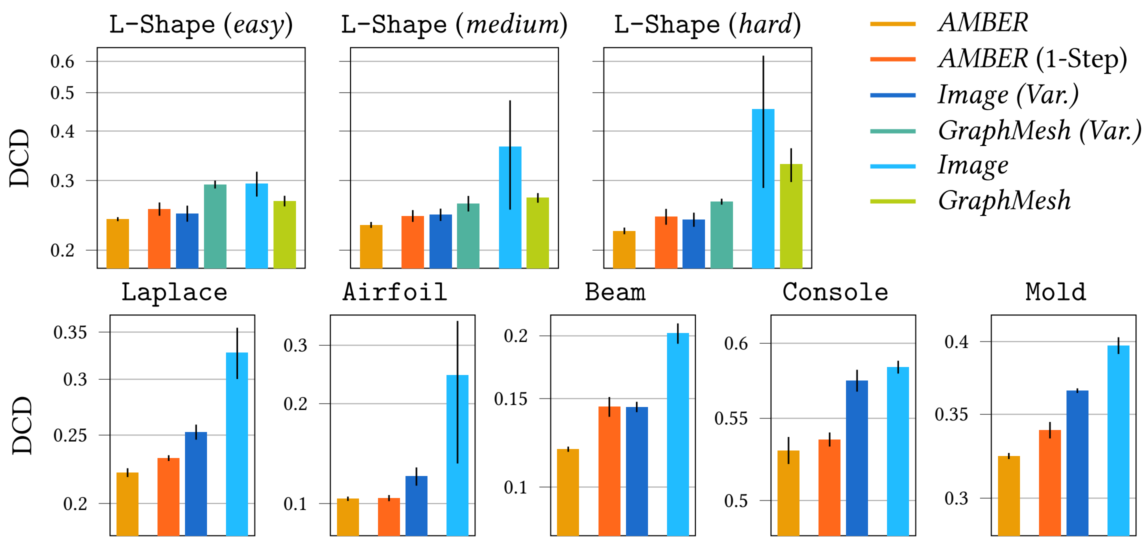


Figure 5.3.: Mean and two times standard error of expert mesh similarity evaluated by DCD (lower is better). *AMBER* achieves the best results across all datasets, demonstrating its ability to generate highly accurate meshes on diverse and challenging domains. All methods perform well on `L-Shape` (*easy*). As dataset complexity increases, the baselines and eventually variants become less reliable. *AMBER (1-Step)* remains strong across datasets, while the full model achieves further improvements through iterative refinement.

L^2 Error Evaluations. To complement the DCD evaluations in Figure 5.3, we additionally assess the L^2 error defined in Equation 5.5 across all datasets and supervised methods in Figure 5.4. While scales are different across datasets, the general trends closely mirror those in Figure 5.3, with only minor differences in relative performance. On the L^2 error, *AMBER* outperforms published baselines on all datasets, and shows a slightly larger advantage over the variants compared to the DCD on, e.g., `L-Shape` and `Beam`. For `Console`, *AMBER (1-Step)* performs well on the L^2 metric, slightly improving over *AMBER*, although error bounds overlap.

Error Indicator Evaluations. We evaluate the norm of the error indicator of Equation 5.6 for `L-Shape` (*hard*) and `Lattice`, i.e., for datasets that use a concrete underlying system of equations. Table 5.1 shows this error indicator norm alongside the number of used mesh elements to account for the norm naturally decreasing with higher element budgets. We find that *AMBER* closely adheres to the element budget of the expert heuristic that was used to generate the data, and that it matches the expert in terms of error indicator. In contrast, many other supervised methods either fail to produce meshes with similar numbers of elements, or have worse error indicator norms, suggesting poor refinements and sub-optimal downstream simulation. These trends highlight *AMBER*'s utility for downstream simulations and support the use of DCD as a reliable proxy on datasets where downstream simulation error is not directly available.

Figure 5.5 compares *AMBER*, *ASMR*, and the expert meshes using the per-mesh norm of the same indicator for `L-Shape` (*easy*/*medium*/*hard*). We obtain Pareto fronts by varying *ASMR*'s element penalty and scaling *AMBER*'s last predicted sizing field at inference between 0.5 and 2.0. All methods follow the expected log-log error-element trend (Dörfler,

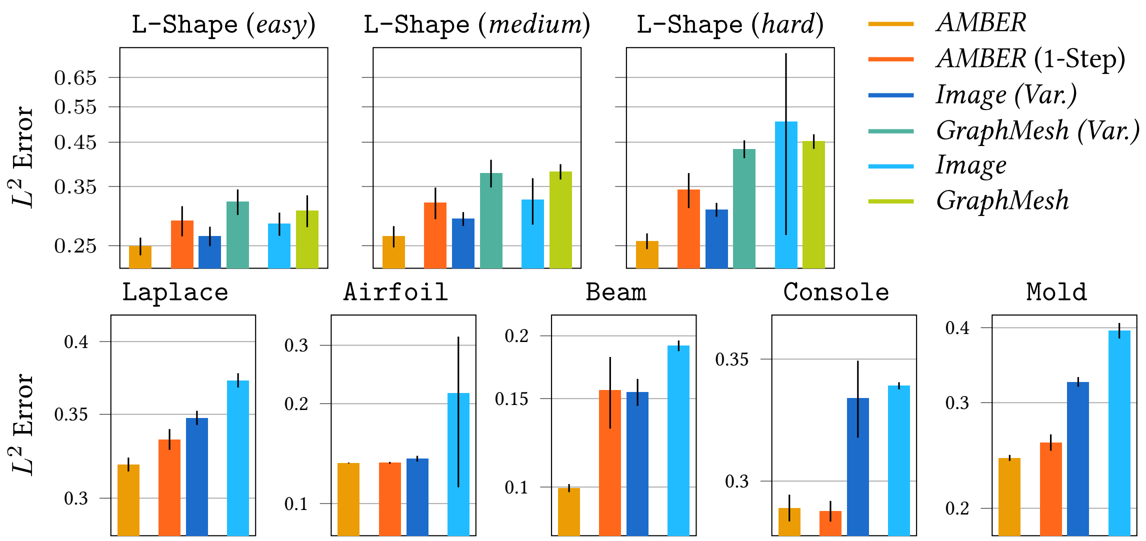


Figure 5.4.: L^2 error across datasets and supervised methods. Overall trends are consistent with Figure 5.3. *AMBER* shows larger relative improvements on datasets like L-Shape and Beam compared to baselines. On Console, *AMBER (1-Step)* slightly outperforms *AMBER*, but with overlapping error bounds.

Table 5.1.: Error indicator norm for L-Shape (*hard*) and Lattice for the expert heuristic and different supervised methods. Overall trends are consistent with Figure 5.3, validating the use of DCD as a proxy for downstream simulation error.

	Poisson		Laplace	
Method	Err. Norm	#Elements	Err. Norm	#Elements
AMBER	0.031 ± 0.001	27859.7 ± 1583.1	2.555 ± 0.050	27622.5 ± 943.1
AMBER (1-Step)	0.032 ± 0.001	28780.9 ± 2196.8	2.568 ± 0.039	27488.7 ± 706.0
Image (Var.)	0.034 ± 0.001	24836.3 ± 1213.0	2.697 ± 0.062	26745.2 ± 866.7
Image	0.082 ± 0.071	130571.3 ± 119228.3	3.235 ± 0.174	29297.8 ± 8065.7
GraphMesh (Var.)	0.042 ± 0.007	46841.7 ± 15014.0	–	–
GraphMesh	0.034 ± 0.001	31378.2 ± 4776.3	–	–
Expert	0.033	25625.2	2.766	25130.5

1996). Markers show test-set averages per target resolution and random seed. *ASMR* exhibits high variance across seeds and degrades beyond ∼30 000 elements due to its fixed-depth reference mesh. In contrast, *AMBER* closely matches and slightly surpasses expert performance on fine meshes, likely due to smoother mesh generation. It also generalizes to >100 000 elements, even though the largest expert mesh has only 31 510 elements. This generalization only requires adjusting a single scalar, enabling zero-shot, budget-aware mesh generation without retraining.

5.5.2. Runtime

We explore *AMBER*'s runtime behavior across different mesh granularities by training on L-Shape (*easy*/*medium*/*hard*) datasets. We evaluate each trained model by setting the last

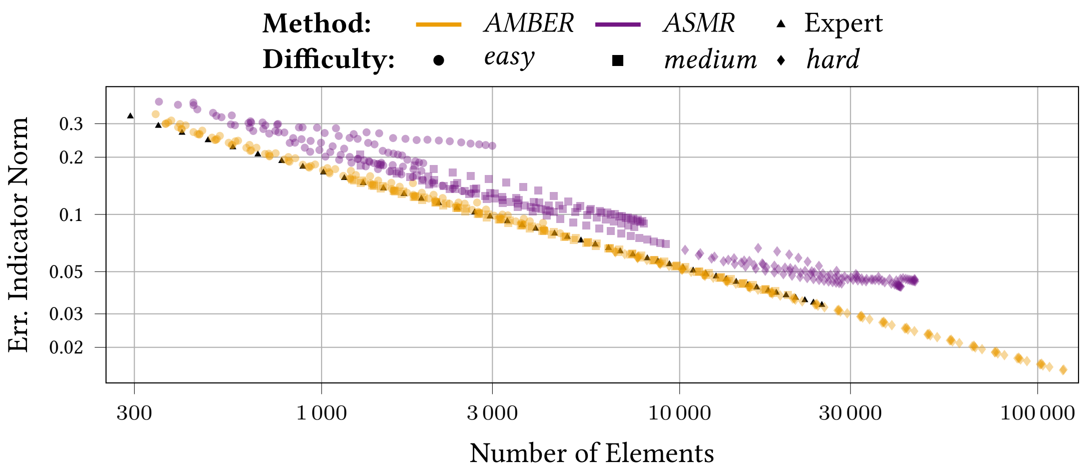


Figure 5.5.: Log-log plot of error indicator norm versus number of mesh elements (lower left is better) for *AMBER*, *ASMR* and the expert across `L-Shape` (*easy*, *medium*, *hard*). Each marker shows the mean over the test set for a given seed. *AMBER* and *ASMR* evaluations are obtained by scaling the final predicted sizing field and tuning the element penalty, respectively. *AMBER* closely matches or even exceeds expert performance in terms of indicator error, and generalizes to meshes that are more than $3\times$ finer, maintaining the expected error-element trend beyond 100 000 elements.

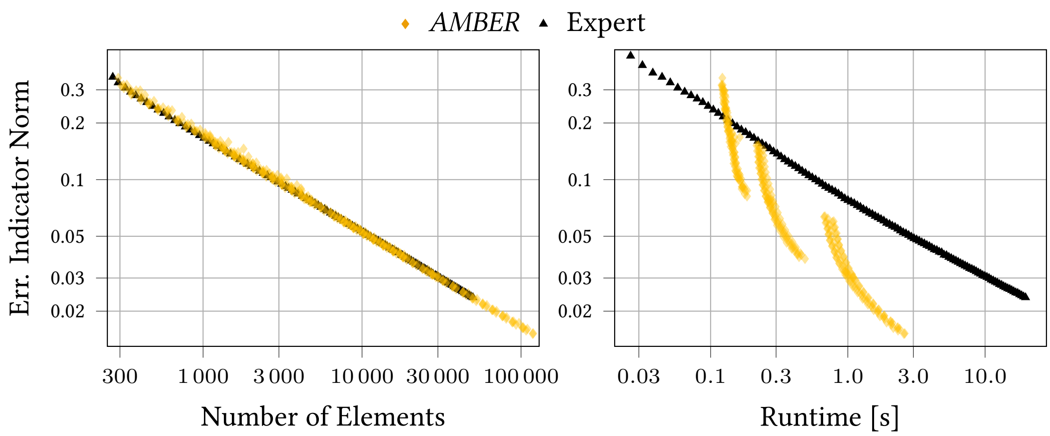


Figure 5.6.: Log-log plot of error indicator norm versus number of mesh elements (**left**) and runtime (**right**) for *AMBER* and the expert across `L-Shape` (*easy*/*medium*/*hard*). Lower left is better. Each marker shows the mean over the test set for a given seed and target mesh resolution. (**Left**) As in Figure 5.5, *AMBER* achieves comparable error to the expert heuristic for a given element budget. (**Right**) For a given training dataset, i.e., any of `L-Shape` (*easy*/*medium*/*hard*), *AMBER* produces roughly the same intermediate meshes, only adapting to the element budget via the scaling constant $c_T \in [0.5, 2.0]$ at the last step. This process causes a distinct runtime curve for each training dataset. *AMBER* scales better with the element budget than the expert heuristic, eventually achieving a speed-up of more than $10\times$ for meshes with more than 30 000 elements.

step's scaling constant $c_T \in [0.5, 2.0]$, as also done in Figure 5.5. Figure 5.6 compares the error indicator norm against both the number of mesh elements and the total runtime for *AMBER* and the expert heuristic. Since the scaling constant only comes into effect at the last mesh generation step, the training dataset significantly influences runtime, with distinct curves for models trained with `L-Shape` (*easy*/*medium*/*hard*). *AMBER* attains

Table 5.2.: Runtime breakdown of `L-Shape` (*easy*/*hard*) in milliseconds. Mesh generation is the most expensive step, and becomes more costly as the number of mesh elements increases.

Category	`L-Shape` (*easy*)		`L-Shape` (*hard*)	
	Mean runtime (ms)	% of total	Mean runtime (ms)	% of total
Mesh to graph conversion	15.815	8.91	94.620	8.37
Adding hierarchical graph	11.219	6.32	12.606	1.11
Model forward	59.963	33.80	155.915	13.79
Mesh generation	90.406	50.96	867.760	76.73

Table 5.3.: Generalization across datasets. Comparison between *AMBER* trained individually per dataset and *AMBER* (*Mixed*) trained jointly on all datasets. The mixed model achieves nearly identical performance, indicating strong generalization and potential for multi-task learned mesh generation.

Method	`L-Shape` (*hard*)	`Lattice`	`Airfoil`
AMBER	0.224 ± 0.004	0.222 ± 0.003	0.103 ± 0.002
AMBER (*Mixed*)	0.226 ± 0.011	0.222 ± 0.005	0.102 ± 0.002

an error comparable to the expert heuristic for all element budgets. However, *AMBER*'s runtime scales significantly better with larger numbers of elements. For large meshes, *AMBER* eventually outperforms the heuristic by more than an order of magnitude in terms of required wallclock time, taking less than 3 seconds to generate a mesh with more than 100 000 elements. We similarly find that *AMBER* takes less than 5 seconds to accurately imitate a 3D mesh on both `Console` and `Mold`, where a human expert needs roughly 15 to 20 minutes for refinement.

Table 5.2 breaks down *AMBER*'s runtime into its main components on `L-Shape` (*easy*/*hard*) for $c_T = 1$. We find that mesh generation quickly dominates runtime, taking up more than $50\,\%$ of total cost for `L-Shape` (*easy*) and jumping to more than $75\,\%$ for `L-Shape` (*hard*). This relative increase in cost is explained by the O($N \log N$) scaling of the mesh generation step, which outscales the linear graph-related operations, including the MPN forward, especially for finer meshes. Notably, *AMBER* acts on coarse intermediate meshes, and the expensive last generation step is also required for the one-step baselines.

5.5.3. Cross-Dataset Generalization

Table 5.3 explores *AMBER*'s ability to train on multiple datasets at the same time. *AMBER* (*Mixed*) shows approximately equal performance to *AMBER* on all considered datasets, opening interesting avenues for multi-task and general-purpose learned mesh generation algorithms in future work.

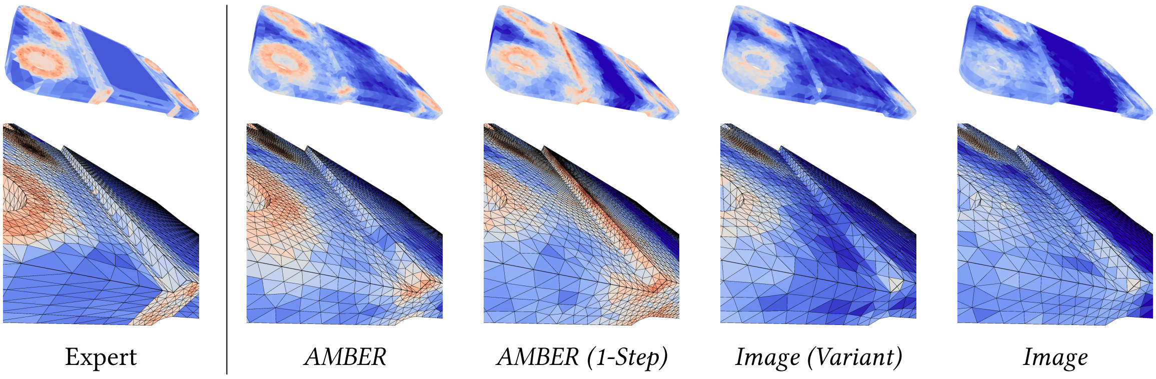


Figure 5.7.: Full views and close-ups of generated `Mold` test meshes. The element size is denoted by color, with red indicating small elements. *AMBER* closely matches the expert mesh, producing finer elements near the hole and coarser elements near the mesh's border. In comparison, the *Image* baselines have less variation in the element size, matching the expert less closely.

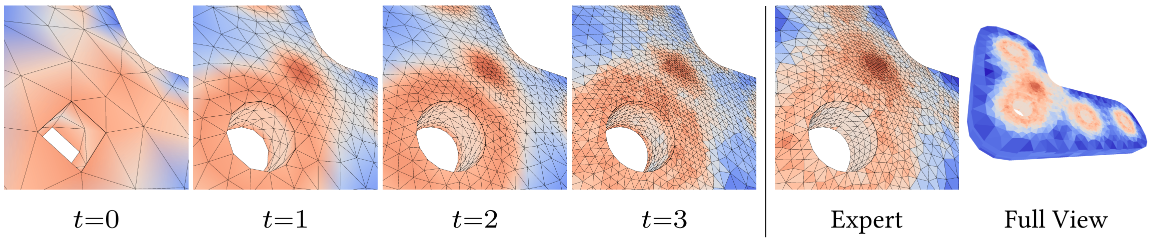


Figure 5.8.: Close-ups of intermediate and final *AMBER* meshes on `Mold`, contrasted with the expert mesh. The color for intermediate meshes denotes the predicted sizing field (red is small), which is given to a mesh generator to produce the next mesh. The final mesh's color denotes its element size.

5.5.4. Qualitative Results

Figure 5.2 shows a final *AMBER* mesh per dataset and Figure 5.7 shows a close-up of generated meshes for different supervised methods on `Console`. Both figures show that *AMBER* produces accurate sizing fields on diverse domains and geometries and produces high-quality meshes, closely resembling the expert. In contrast, the *Image* baselines only learn general, low-frequency features of the expert's sizing field, but fail to capture finer details. The result is a comparatively more uniform, less adaptive mesh. Figure 5.8 provides a full *AMBER* rollout on `Console`, showcasing the iterative generation process. In each step, *AMBER* consumes the previous mesh, using it to predict an increasingly accurate sizing field. We provide further visualizations for *AMBER* rollouts and generated meshes for all baselines in Section C.5.

5.6. Parameter Study and Further Analysis

We explore and validate the design choices of *AMBER* and our baselines through several additional experiments. For *AMBER*, we explore the loss in Equation 5.2 and the components from Section 5.3.3. We also explore alternative sizing field parameterizations for $\hat{f}$,

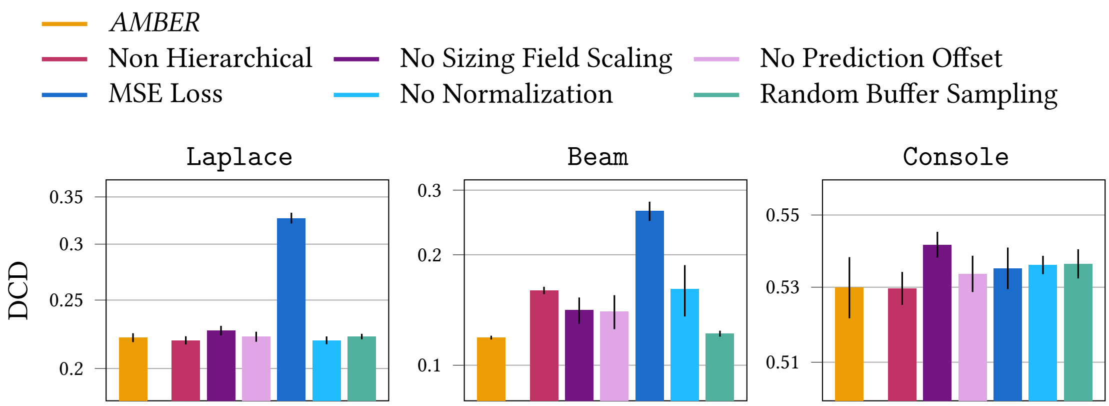


Figure 5.9.: Design study on *AMBER* using Density-Aware Chamfer Distance (DCD) across three datasets. Each bar represents a variant of the model with one component removed or modified. Using a non-hierarchical MPN, omitting the prediction offset, or sampling newly generated meshes randomly degrades performance moderately, depending on the dataset. Omitting normalization or using a regular MSE loss leads to substantially worse generated meshes.

including a different piecewise-linear sizing field and a piecewise-constant option. Finally, we vary *AMBER*'s training data size and its number of mesh generation steps. For the *Image (Variant)* baseline, we explore lower input resolutions and versions that omit either the loss or input/output normalization. We additionally consider an *ASMR* variant that uses the error indicator as reward.

5.6.1. Algorithmic Design Choices

We evaluate the importance of several of *AMBER*'s components on `Lattice`, `Beam` and `Console`. To evaluate the impact of the loss of Equation 5.2, we compare against an *AMBER (MSE Loss)* variant using a direct MSE loss between the softplus-transformed predictions and the sizing field targets, i.e.,

$$\mathcal{L} = \frac{1}{|V|}\sum_{j=1}^{V}(\log(1+e^{x_j}) - y_j)^2.$$

For the algorithmic components, we first replace the stratified sampling for the replay buffer with uniform sampling over all intermediate meshes (*AMBER (Random Buffer Sampling)*). This results in an over-representation of meshes with many prior refinements, leading to a skewed and unbalanced training distribution. Next, we disable the hierarchical mesh representation, feeding only the non-hierarchical graph $\mathcal{G}$ into the MPN (*AMBER (Non-hierarchical)*). This reduces consistency in the receptive field across and within meshes, as regions with higher local resolution require more message passing steps. We also ablate the normalization (*AMBER (No Normalization)*) and the offset term b_j of Section 5.3.3 (*AMBER (No Prediction Offset)*). Finally, we remove the scaling of sizing fields for intermediate meshes by setting the refinement constant of Section 5.3.3 to c_t=1 for all t (*AMBER (No Sizing Field Scaling)*). While this does not directly impact optimization, it significantly

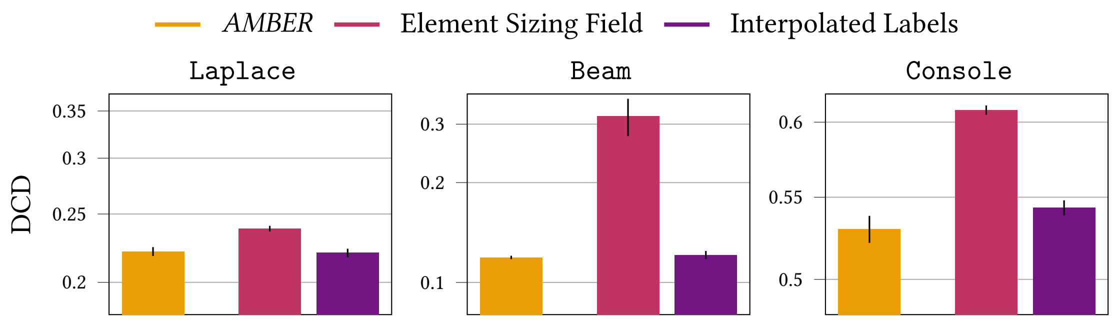


Figure 5.10.: DCD comparison across datasets for different sizing field parameterizations. Interpolating the labels closely matches *AMBER*'s parameterization, reflecting similar optimization objectives. In contrast, using a piecewise-constant sizing field on the elements yields worse meshes on `Beam` and `Console`, likely due to reduced expressiveness.

increases intermediate mesh sizes, slowing down mesh generation during training and inference, and reducing the number of meshes that fit in a training batch.

Figure 5.9 presents the results of aforementioned algorithmic variants. *AMBER* consistently performs on par with or better than its variations across all datasets. Replacing our loss with a regular MSE leads to the largest degradation in performance, consistently yielding worse meshes than *AMBER* across datasets. Depending on the dataset, different algorithmic components have different impact. The hierarchical graph representation is crucial on `Beam`, as it requires long-range message passing to capture the spatial dependencies of the elongated geometry. The softplus-transformed loss is essential for `Lattice` given its high element scale variation. The sizing field scaling only improves mesh quality slightly, but decreases the size of intermediate meshes, speeding up training and inference. Other factors like normalization and buffer sampling have smaller effects, but still generally yield modest benefits.

5.6.2. Sizing Field Parameterization

We experiment with different parameterizations of the predicted sizing field on `Lattice`, `Beam` and `Console`. Given an expert mesh M^*, we consider using the vertex-interpolated expert sizing field $f(v_j)$, as defined in Equation 5.3 to define labels $y_j = \mathcal{I}_{M^*}(f)(\mathbf{p}(v_j))$ using the interpolant of Figure 5.6.2. We call this variant *AMBER (Interpolated Labels)*.

Additionally, we consider a version that predicts a piecewise-constant sizing field $\hat{f}_e(M_i)$ on the elements M_i instead of a piecewise-linear sizing field $\hat{f}(v_j)$ on the vertices v_j. Here, the corresponding interpolant $\mathcal{I}_M(f_e)$ is just the union over the element's predictions evaluated at their subdomain, i.e.,

$$\mathcal{I}_M(f_e)(\mathbf{z}) = \begin{cases} f_e(M_i), & \text{if } \mathbf{z} \in \Omega_i \text{ for some } i, \\ f_e(M_{i'}), & \text{otherwise, where } i' = \arg\min_i \|\mathbf{z} - \mathbf{p}(M_i)\|, \end{cases}$$

where $\mathbf{p}(M_i)$ denotes the position of the element's midpoint. We assign each element the integrated sizing field of all expert elements that it contains, i.e., we compute a volume-weighted average of the sizing field values from the fine mesh elements whose midpoints lie within the coarse element. Let $f_e^*(M_k^*)$ be the sizing field on the fine elements and $\text{Vol}(M_k^*)$ their volume. For each element M_i of the current mesh, we compute the target sizing field as

$$y_i = \begin{cases} \sum_{k \in \mathcal{J}_i} \frac{\text{Vol}(M_k^*)}{\sum_{k \in \mathcal{J}_i} \text{Vol}(M_k^*)} f_e(M_k^*), & \text{if } \mathcal{J}_i \neq \emptyset, \\ f_e(M_k^*), & \text{if } \mathbf{p}(M_i) \in \Omega_k^* \text{ for some } M_k^*, \\ f_e(M_{k'}^*), & \text{otherwise,} \end{cases}$$

where $k' = \arg\min_k \|\mathbf{p}(M_i) - \mathbf{p}(M_k^*)\|$ and $\mathcal{J}_i = \{j \mid \mathbf{p}(M_k^*) \in M_i\}$ is the set of expert elements whose midpoints lie within the element M_i. If there are no such elements, we first attempt to find an expert element $M_{j'}$ that contains the midpoint of M_i. If that also fails, the meshes represent different discretizations of the underlying domain. Here, we fall back to nearest-neighbor interpolation using the element midpoint positions. We adapt the MPN input accordingly, constructing the graph $\mathcal{G}$ over mesh elements and element neighborhood. We use the same graph node and edge features, except for the neighborhood size, and always evaluate position-dependent features at the element midpoint. This process yields a variant *AMBER (Element Sizing Field)*.

Figure 5.10 visualizes results. We find that *AMBER (Interpolated Labels)* closely matches *AMBER*, likely because both optimize a similar objective. While there are small differences in the concrete sizing field targets, especially for early generation steps and coarser input meshes, both parameterizations work well. Both parameterizations provide targets that aim to coarsen too-fine regions, while increasing the resolution in too-coarse regions of the current mesh, eventually converging to very similar generated meshes. In contrast, *AMBER Element Sizing Field* predicts a piecewise-constant sizing field over mesh elements. While this works well on `Lattice`, the reduced expressiveness of this parameterization compared to the piecewise-linear interpolant of Equation 5.1 yields significantly worse meshes on both `Beam` and `Console`.

5.6.3. Data Efficiency

Figure 5.11 assesses *AMBER*'s data efficiency. All other training settings are held constant, and evaluation is performed on the original test set. Accurate mesh generation is achieved with as few as five training meshes and corresponding geometries. Using more samples leads to modest performance improvements, and *AMBER* tends to generalize well from only five training samples. On `Lattice`, where training data can be easily generated via the expert heuristic, there are additional improvements for 100 instead of 20 meshes. *AMBER*'s efficient use of data likely stems from the local, per-node loss in Equation 5.2 and the symmetry-preserving features and structure of the MPN architecture.

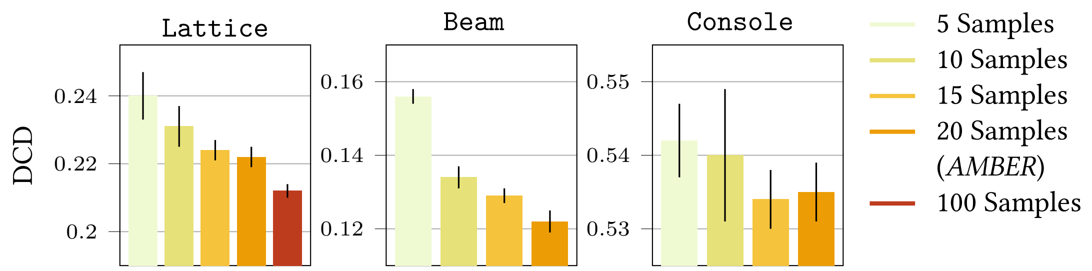


Figure 5.11.: DCD comparison for *AMBER* with different numbers of training samples for `Lattice`, `Beam` and `Console`. *AMBER* performs well with as little as 5 samples, but steadily improves for up to 20 samples. On `Lattice`, where additional training data is easy to generate, there is a moderate improvement for 100 instead of 20 train meshes and geometries.

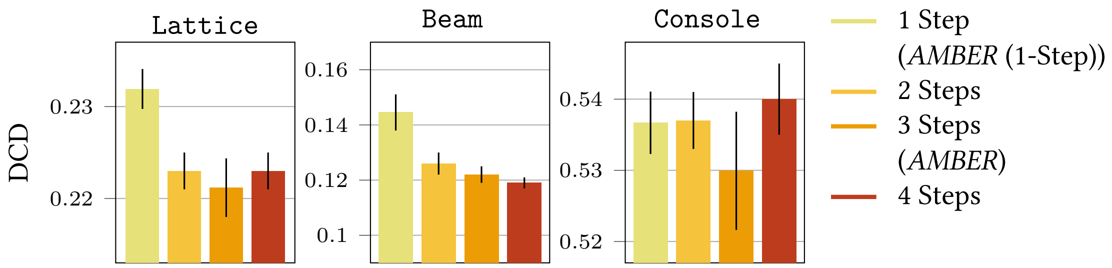


Figure 5.12.: DCD comparison for *AMBER* with different numbers of mesh generation steps for `Lattice`, `Beam` and `Console`. *AMBER* improves for more mesh generation steps, and converges at around three steps. A single mesh generation step is insufficient for accurate generations, likely because it acts on a fixed mesh resolution. Results for *AMBER* and *AMBER* (1-Step) are taken from Figure 5.3.

5.6.4. Mesh Generation Steps

We evaluate how *AMBER* behaves for different numbers of mesh generation steps. In particular, we use three generation steps in all main experiments, and have a single step for *AMBER* (1-Step) as a baseline that acts on a tuned but fixed mesh resolution per dataset. Figure 5.12 shows that *AMBER* generally improves for more mesh generation steps. Despite tuning the initial mesh size, a single mesh generation step is insufficient for optimal performance, presumably because it does not allow for arbitrarily fine sizing field resolution. In contrast, starting from two mesh generation steps, *AMBER* learns to predict the sizing field used to generate its intermediate meshes, allowing for a flexible, adaptive sizing field representation. For most datasets, performance converges at around three mesh generation steps.

5.6.5. Extended Baseline Experiments

Section C.4 provides additional experiments for the *Image* and *ASMR* baselines. Section C.4.1 explores different configurations of *Image (Variant)*. Performance improves with mesh resolution, but the gains diminish at finer scales, likely because increasing the resolution to high-detail regions becomes increasingly expensive. Omitting either our loss function or the input/output normalization degrades performance, showing the importance of both components.

Section C.4.2 introduces an *ASMR (Error Indicator)* variant that uses the error indicator as reward. *ASMR (Error Indicator)* is not constrained by a fine-grained reference mesh and can thus refine to arbitrary resolutions. Experimentally, this version avoids *ASMR*'s degradation on fine meshes but provides significantly less consistent and overall worse refinements, likely due to a weaker reward signal.

5.7. Conclusion

We introduce *AMBER*, a novel method for iterative Adaptive Mesh Generation (AMG) that combines a replay buffer of bootstrapped data with Message Passing Graph Neural Networks operating on intermediate meshes. At each step, *AMBER* consumes the current mesh to predict a target resolution for the next one, allowing fine-grained adaptation to complex geometries. *AMBER* generates high-quality adaptive meshes across six novel datasets spanning diverse and realistic 2D and 3D geometries, consistently outperforming supervised and reinforcement learning baselines. These results demonstrate the effectiveness of learning-based approaches in reducing manual meshing effort, contributing toward more efficient and scalable simulation workflows in engineering applications.

5.7.1. Limitations and Future Work

AMBER predicts scalar-valued, piecewise-linear sizing fields, which limits expressiveness in scenarios requiring extreme variation in local mesh density, or anisotropic refinement. Predicting tensor-valued sizing fields or using higher-order polynomials is a promising direction for future work. Furthermore, our experiments indicate that the same model can be trained on different datasets, showing that there is potential to train a general-purpose model across a vast number of datasets. Lastly, assessing the performance of *AMBER* directly through simulation error metrics on real-world scenarios is an interesting avenue for future research.

5.7.2. Broader Impact

AMBER has the potential to benefit numerous domains that depend on computational modeling and simulation by significantly reducing computational costs without compromising accuracy. This efficiency can expand the scope of feasible simulations in engineering design, and support the deployment of simulation-based tools in resource-constrained environments. Nonetheless, as is common with advanced computational tools, there is a risk of misuse in contexts such as weapons development or unsustainable resource exploitation.

6. Discrete Trust Region Optimization for Large Language Models

Expanding our notion of geometry from physical spaces to the manifold of probability distributions, this chapter addresses policy optimization in high-dimensional action spaces. While the previous chapters utilized geometric representations to facilitate robotic manipulation and adaptive meshing, we now treat the policy's output distribution itself as the primary (information-)geometric object. Concretely, Trust Region Optimization for Large Language models (*TROLL*) addresses the limitations of heuristic clipping in RL for Large Language Models (LLMs) by introducing a principled, differentiable projection onto information-geometric trust regions. In doing so, it answers our third research question, which is restated below.

Q3 How can we employ information-geometric constraints to stabilize policy optimization in high-dimensional action spaces? (Chapter 6)

This chapter serves as the final pillar of this thesis, bridging the gap between the physical geometric reasoning developed in the previous chapters and the implicit information geometry underlying policy updates in modern generative models. *TROLL* views policy updates as movements within a constrained distribution space by extending the differentiable trust region projections introduced in Section 2.2 to the discrete LLM vocabularies. This extension replaces PPO-style clipping, which we introduce in Section 2.1.2 as a crude first-order approximation of a trust region, with a more principled optimization problem. As discussed in Section 2.1.2, such a shift is particularly relevant for LLMs given the unique challenges posed by massive vocabularies and the long sequences and resulting observation spaces encountered in reasoning tasks.

To make *TROLL*'s trust region projections computationally viable for the massive vocabularies of modern LLMs, we introduce a sparsification scheme based on the observation that language distributions are typically heavy-tailed. By preserving only the most relevant tokens for each policy update, we reduce the dimensionality of the optimization problem without sacrificing the stability of the trust region. We evaluate *TROLL* across a variety of mathematical reasoning and coding benchmarks, applying it to several models and advantage-estimation methods. These experiments demonstrate the impact of trust region projections on training stability, convergence speed and final performance. Additionally, we demonstrate the versatility of this optimization framework by combining it with *ASMR* from Chapter 4 to stabilize AMR decisions.

The following work was published as **TROLL: Trust Regions improve Reinforcement Learning for Large Language Models** *(Philipp Becker*, Niklas Freymuth*, Serge Thilges,* *Fabian Otto, Gerhard Neumann) in 14th International Conference on Learning Representations, ICLR 2026 (Oral). Reprinted with permission of the authors. Wording, notation, structure and formulations were revised in several places.*

6.1. Introduction

RL has become the standard approach for fine-tuning and aligning LLMs with preferences or verifiable rewards. For such post-training, the algorithms of choice are predominantly PPO-style policy gradient approaches (Schulman et al., 2017). They first estimate an advantage function and then update the policy using an importance-weighted objective, clipped to prevent the ratio between new and old policies from deviating too much. Recent approaches such as Group-Relative Policy Optimization (GRPO) (Shao et al., 2024), GRPO Done Right (Dr.GRPO) (Liu et al., 2025), Group Sequence Policy Optimization (GSPO) (Zheng et al., 2025), and REINFORCE++ (Hu et al., 2025) improve the estimation of advantages and normalization, resulting in significant advances in RL for LLMs. Yet, all these approaches rely on PPO's clipping-based policy update mechanism.

Clipping is originally motivated via trust region methods (Schulman et al., 2015a; Schulman et al., 2017), which provide a principled way to stabilize policy updates by constraining the KL divergence (Kullback et al., 1951) between successive policies during training (Kakade et al., 2002b; Peters et al., 2010). While theoretically sound, implementing such trust regions is often costly, in particular with modern LLMs whose vocabularies and thus output distributions can exceed 100 000 entries (Yang et al., 2025a; Yang et al., 2025b). PPO instead clips the importance ratio to sidestep this challenge. While empirically successful, it is a crude approximation of the underlying trust region (Wang et al., 2019; Wang et al., 2020b). Crucially, it may lead to unstable optimization, poorly calibrated updates, as well as sensitivity to hyperparameters and implementation details, which often culminate in suboptimal performance (Engstrom et al., 2020; Andrychowicz et al., 2021; Otto et al., 2021; Huang et al., 2022).

As a remedy, we introduce Trust Region Optimization for Large Language models (*TROLL*), a differentiable trust region projection approach that directly enforces token-level KL constraints between discrete distributions. *TROLL* formulates a convex optimization problem that acts as a direct replacement to PPO-like clipping objectives. For each token, *TROLL* projects the output distribution of the new, updated policy onto a KL-trust region around the old policy that was used to sample the sequence. This process ensures that the new and old policies only differ by a given bound, preventing the policy update from diverging or collapsing. The left side of Figure 6.1 shows a 3-dimensional example where the old policy prefers the ”troll” token, the new policy leans toward the ”hamster” token, and the trust region constrains the update to keep the new policy close to the old one. We show that the direction of the projection can be computed in closed form, while its step

* Equal contribution
Project page, code, and datasets available at `https://niklasfreymuth.github.io/troll/`

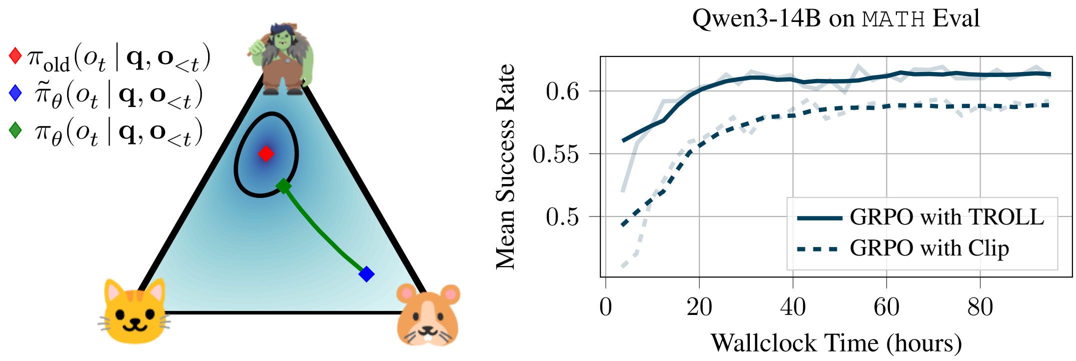


Figure 6.1.: *TROLL* overview. (**Left**) Example of a 3-token distribution (cat, troll, hamster). The old policy (red) favors the troll, while the new policy (blue) shifts toward the hamster. The projection (green) ensures that the updated policy stays within the trust region (black). (**Right**) This projection yields clear performance gains over PPO clipping on our `Math`-Eval suite (see Section 6.4), as shown here for Qwen3-14B trained with GRPO.

size can be efficiently computed by solving a one-dimensional convex Lagrangian dual problem. Crucially, the projection leaves the new distribution unchanged if it already falls within the trust region, and can be solved and parallelized efficiently in practice. *TROLL* enables differentiation through the solution of the projection problem using the OptNet framework (Amos et al., 2017), which introduces only negligible computational overhead. Differentiating through the projection allows *TROLL* to maintain gradient information even for updates that are constrained by the trust region, in contrast to PPO-like clipping, which cuts gradients for tokens whose ratios exceed the clipping threshold. Further, the trust region is only effective during training and imposes zero additional overhead during model inference. To incentivize the model to stay within the trust region for successive update steps, we additionally add a simple regression term between projected and unprojected tokens.

Applying *TROLL* directly to LLMs is computationally infeasible, since the large token vocabulary causes prohibitively expensive projections and memory overhead. However, natural language continuations and similarly LLM predictions are generally characterized by very few high-probability tokens (Zipf, 1949; Piantadosi, 2014; Kunstner et al., 2024; Duan et al., 2024; Ren et al., 2024). This property lets us introduce a sparsification scheme that discards the vast majority of low-probability tokens, retaining only the most relevant ones. We find that, on average, as few as $5-10$ tokens generally preserve more than 99.999% of the distribution's probability mass. We thus modify our differentiable trust region projection to handle sparse distributions, allowing them to scale to modern LLMs and act as a drop-in replacement for PPO-style clipping.

We evaluate *TROLL* on Reinforcement Learning from Verifiable Rewards (RLVR) problems, focusing on mathematical reasoning. Using *TROLL* for GRPO (Shao et al., 2024) with models from the Qwen3 (Yang et al., 2025a) and Qwen2.5 (Yang et al., 2025b) families on DAPO-Math (Yu et al., 2025) yields substantial improvements in terms of training stability and success rate for evaluations based on the number of updates as well as wall clock time. Concretely, *TROLL* improves roughly $3-10$ percentage points, or $5-15\%$ relative over clipping for math-based reasoning. To assess robustness across algorithmic variants, we experiment with PPO (Schulman et al., 2017), Dr.GRPO (Liu et al., 2025), GSPO (Zheng et al., 2025), and REINFORCE++ (Hu et al., 2025). Across methods, *TROLL* consistently enables faster learning and improves success rates, indicating that its benefits are independent of the underlying advantage estimation method. We demonstrate similar improvements across additional math datasets, namely GSM8K (Cobbe et al., 2021b) and Eurus-2-RL-Math (Cui et al., 2025a), as well as models from the LLaMA 3 (Grattafiori et al., 2024), SmolLM3 (Bakouch et al., 2025), and Apertus (Hernández-Cano et al., 2025) families. Here, *TROLL* enables stable training for some models where clipping fails entirely. Finally, we consider code generation using the Eurus-2-RL-Code (Cui et al., 2025a) dataset, showing improvements of $7-18$ percentage points, or $18-30\%$ relative over clipping across Qwen3 model sizes.

We summarize our contributions as follows: **(i)** we derive Trust Region Optimization for Large Language models (*TROLL*), a fully differentiable, principled trust region projection for discrete distributions that enforces per-token KL constraints, **(ii)** we introduce a sparsification scheme that makes the projection scale to large vocabularies and implement it as a drop-in replacement for PPO-style heuristic clipping across RL algorithms, **(iii)** we demonstrate through extensive experiments spanning different advantage-estimation methods, models and datasets that *TROLL* consistently improves both reward and training stability compared to clipping.

6.2. Related Work

Trust Regions in Reinforcement Learning. Information-theoretic trust regions based on the KL divergence (Kullback et al., 1951) are known to stabilize RL in classical (Kakade, 2001; Kakade et al., 2002b; Peters et al., 2010; Abdolmaleki et al., 2015; Akrour et al., 2018) as well as modern deep learning settings (Schulman et al., 2015a; Schulman et al., 2017). Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a) limits the KL-divergence by solving a constrained optimization problem. PPO (Schulman et al., 2017) simplifies this approach to a first-order method, using a clipped surrogate objective to make policy optimization in RL scalable (Akkaya et al., 2019; Berner et al., 2019; Baker et al., 2020). However, PPO's trust region is less principled and very sensitive to practical implementation details (Engstrom et al., 2020; Andrychowicz et al., 2021; Huang et al., 2022). Several methods dynamically adapt PPO's clipping bounds (Wang et al., 2019; Xi et al., 2025), but still require clipping. Alternative methods (Abdolmaleki et al., 2018; Song et al., 2020) rely on regularizing the expected KL between subsequent policies. In contrast, *TROLL* enforces exact, token-wise trust regions through a differentiable KL projection, combined with adaptive strategies that control the trust region size.

Trust Region Projections. A different line of work (Otto et al., 2021; Akrour et al., 2019) enforces trust regions using projection-based methods. These approaches first compute the policy as usual, and then project it back into a feasible set defined by a trust region constraint. In particular, Otto et al. (2021) compute exact trust region projections for each state when using Gaussian policies, leading to improvements in high-dimensional action spaces (Celik et al., 2024; Li et al., 2024a; Hoang et al., 2025; Otto et al., 2025). Similar to us, they rely on Lagrangian optimization (Boyd et al., 2004) and implicit differentiation (Amos et al., 2017), but focus on Gaussian distributions and continuous control tasks (Brockman et al., 2016). *TROLL* instead proposes differentiable projections for categorical distributions and provides an efficient implementation via a sparsification scheme. This combination lets *TROLL* scale to modern-day LLMs while preserving the stability of classical trust region methods.

Reinforcement Learning for LLMs. Recently, RL has become a key tool in the post-training stage of LLMs. Popular frameworks include RL from human feedback (RLHF) (Christiano et al., 2017; Ziegler et al., 2019; Stiennon et al., 2020; Ouyang et al., 2022) for LLM alignment and RLVR (Trung et al., 2024; Lambert et al., 2024) for reasoning tasks such as mathematical problem solving or code generation. In RLHF, preference-based methods often optimize a reward while remaining close to a reference policy via an added expected KL penalty. This reference policy is typically the supervised fine-tuning model (Stiennon et al., 2020; Ouyang et al., 2022). For example, Direct Preference Optimization (DPO) (Rafailov et al., 2023) optimizes this objective in closed form on preference data, thereby avoiding policy rollouts. *TROLL* does not constrain the updates to a fixed reference, instead enforcing proximity to the previous step's policy to stabilize on-policy optimization. While this work focuses on RLVR for LLMs, *TROLL* can also be applied to other LLM post-training settings, and more generally, discrete RL tasks.

Several recent methods avoid PPO's importance sampling and thus clipping entirely. These methods combine group rewards with a standard policy gradient objective (Chu et al., 2025), utilize an additional value network (Richemond et al., 2024), use Q-learning (Clavier et al., 2025), or employ a contrastive loss to enable off-policy learning (Flet-Berliac et al., 2024; Cohere et al., 2025). However, while these approaches are promising, current research on RLVR still often relies on advantage estimation. While not the focus of this work, we briefly show that one such non-clipping-based approach also benefits from *TROLL*, since its trust regions directly act on the LLM's policy.

Advantage Estimation for LLM Post-Training. PPO (Schulman et al., 2017) requires an expensive explicit value model. Especially for RLVR, where evaluating multiple rollouts per input is comparatively cheap, sample-based advantage estimation methods like Group-Relative Policy Optimization (GRPO) (Shao et al., 2024) have become popular alternatives. Variants include GRPO Done Right (Dr.GRPO) (Liu et al., 2025), which addresses GRPO's optimization biases that favor longer responses, and REINFORCE++ (Hu et al., 2025), which uses Global Advantage Normalization (Andrychowicz et al., 2021) to improve training stability. Group Sequence Policy Optimization (GSPO) (Zheng et al., 2025) extends importance ratios from token to sequence level to improve update stability. However, all these methods still depend on PPO-style clipping to stabilize policy updates. We introduce *TROLL* as a more principled drop-in replacement, applicable regardless of how advantages are computed and compatible with all the approaches above.

6.3. Trust Region Optimization for Large Language Models

RL for LLMs usually finetunes the model parameters θ by employing policy ratio objectives (Schulman et al., 2015a). Using the LLM-specific notation of Section 2.1.2, these take the form

$$\mathcal{J}_{\text{ratio}}(\pi_\theta) = \mathbb{E}_{\mathbf{o}\sim\pi_{\text{old}}(\mathbf{o}|\mathbf{q})\mathcal{D}(\mathbf{q})}\left[\frac{1}{|\mathbf{o}|}\sum_{t=1}^{|\mathbf{o}|}\left(\frac{\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}A(o_t, \mathbf{q}, \mathbf{o}_{<t})\right)\right], \tag{6.1}$$

where $\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ is the probability of the sampled token under the current LLM policy. Here, $\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ is the token's probability under the LLM policy that was used for data collection in the previous iteration. The context sequence consists of the prompt $\mathbf{q}$ and prior response tokens $\mathbf{o}_{<t}$. The advantage estimate $A_t = A(o_t, \mathbf{q}, \mathbf{o}_{<t})$ measures if a token is better or worse than the average behavior. Thus, maximizing $\mathcal{J}_{\text{ratio}}(\pi_\theta)$ increases the probability of good responses while decreasing the probability of bad ones. In practice, we estimate A_t using an explicit value model as in PPO (Schulman et al., 2017) or purely sample-based as in GRPO and its variants (Shao et al., 2024; Liu et al., 2025; Zheng et al., 2025). For such policy ratio objectives, stable and effective optimization requires keeping $\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ and $\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ close, so that the importance ratio

$$\omega_t(\theta) = \frac{\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}$$

remains close to one (Schulman et al., 2015a; Schulman et al., 2017). PPO attempts to maintain this proximity by clipping the ratio,

$$\mathcal{J}_{\text{ppo}}(\pi_\theta) = \tag{6.2}$$
$$\mathbb{E}_{o_t\sim\pi_{\text{old}}(\mathbf{o}|\mathbf{q})\mathcal{D}(\mathbf{q})}\left[\frac{1}{|\mathbf{o}|}\sum_{t=1}^{|\mathbf{o}|}\min\left(\omega_t(\theta)A_t; \text{clip}\left(\omega_t(\theta), 1-\epsilon_{\text{ppo}}, 1+\epsilon_{\text{ppo}}\right)A_t\right)\right].$$

However, this clipping is a crude approximation of the underlying trust region principle. While it prevents large updates, this approach is purely heuristic and suppresses gradients when the ratio falls outside the clipping range, resulting in unstable and inefficient learning. In contrast, token-wise KL-based constraints offer a principled approach to limit the change between successive policies. Our method, Trust Region Optimization for Large Language models (*TROLL*), implements these constraints using differentiable trust region projections (Otto et al., 2021) as a drop-in replacement for the PPO-like clipping.

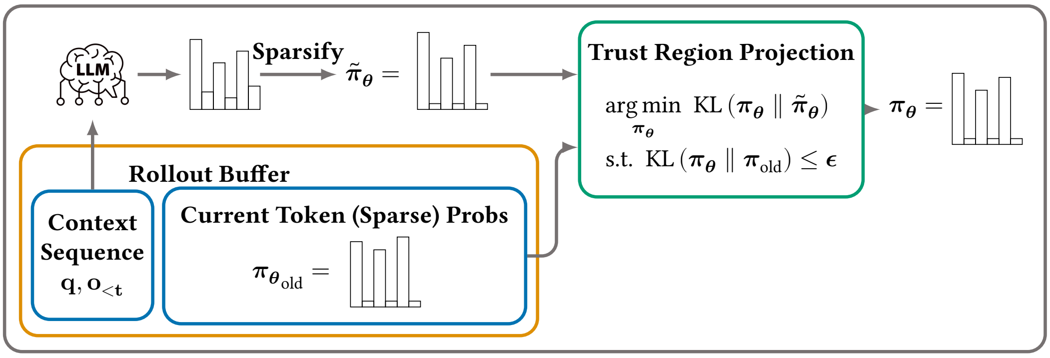


Figure 6.2.: *TROLL* replaces PPO-like clipping with a differentiable trust region projection approach. Given the current output distribution of an LLM $\tilde{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$, and the distribution that was used to collect the sequence $\pi_{\text{old}}(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$ for the replay buffer, *TROLL* enforces a per-token trust region by solving Equation 6.3 for each token. The resulting distribution is then used in the RL objective (Equation 6.5) to update the LLM parameters. To scale this approach to the vocabulary size of modern LLMs, *TROLL* uses a sparsification approach, which allows working on a small subset of logits while retaining most of the distribution's mass.

6.3.1. Discrete Differentiable Trust Region Projections

Trust Region Projection. Our projection formally solves the convex optimization problem

$$\pi_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t}) = \underset{\hat{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})}{\arg\min} \; \mathrm{KL}\left(\hat{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t}) \,\|\, \tilde{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})\right) \quad (6.3)$$
$$\text{s.t. } \mathrm{KL}\left(\hat{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t}) \,\|\, \pi_{\text{old}}(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})\right) \leq \epsilon$$

for every output token o_t. Intuitively, the projection finds the policy distribution closest to the current LLM policy $\tilde{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$ while remaining within an ϵ-bound of the policy used to collect the data for the current iteration $\pi_{\text{old}}(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$. The solution to this optimization problem is derived in Section D.1.1 and given as

$$\pi_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t}) \propto \exp\left(\frac{\eta^* \log \pi_{\text{old}}(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t}) + \log \tilde{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})}{\eta^* + 1}\right), \quad (6.4)$$

which is a geometric interpolation between the logits of $\tilde{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$ and $\pi_{\text{old}}(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$. Here, η^* acts as a step size controlling how far the projection moves the new policy to the old one. For each token, we can compute the optimal η^* which enforces the trust region constraint by solving the convex dual of Equation 6.3. This dual is a scalar optimization problem, which we derive and state in Section D.1.2, and can be solved with sufficient accuracy using a few iterations of ternary, or more generally, n-ary, bracketing. Furthermore, projecting is only necessary if the trust region bound is violated, which is only the case for very few, but highly relevant tokens. Thus, we can avoid it for the vast majority of tokens by filtering them beforehand.

$\pi_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$ must also remain a valid distribution, i.e., $\sum_{o_t} \pi_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t}) = 1$ and $\pi_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t}) \geq 0$ for all o_t. We omit these constraints for brevity and elaborate in Section D.1.

Policy Updates with Trust Region Projections. The policy $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ satisfies the trust region constraint after the projection and can be used to optimize the LLM parameters θ, via an update similar to Equation 6.1. However, the LLM output $\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ may still deviate arbitrarily from the old policy, complicating inference and successive updates. To avoid this issue, we follow Otto et al. (2021) and address this by regressing the LLM output $\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ toward its projection $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$, resulting in an objective

$$\mathcal{J}_{\text{Troll}}(\pi_\theta) = \tag{6.5}$$
$$\mathbb{E}\left[\frac{1}{|\mathbf{o}|}\sum_{t=1}^{|\mathbf{o}|}\left(\frac{\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}A_t\right) - \alpha \text{KL}\left(\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) \parallel \lfloor \pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) \rfloor\right)\right],$$

where the expectation is over tokens $o_t \sim \pi_{\text{old}}(\mathbf{o}|\mathbf{q})\mathcal{D}(\mathbf{q})$, $\lfloor \ \rfloor$ denotes the stop gradient operator and α is a user-specified regression weight. Crucially, the projected policy $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ is used to compute the ratios and as a regression target for the LLM output $\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$. For the regression, we stop the gradients through $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ so that the LLM policy $\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ is pulled toward the output of the projection $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$, not the other way around. The regression term only affects projected tokens and still allows policy updates up to the KL bound, making the approach robust to the choice of α. We thus set $\alpha = 1$ in all experiments for simplicity. Notably, our objective in Equation 6.5 makes no assumption on the advantages A_t. Thus, *TROLL* can be directly applied to a variety of existing advantage estimation methods, including PPO, GRPO, Dr.GRPO, and GSPO. algorithm 4 provides pseudocode.

Making Trust Region Projections Differentiable. To propagate gradients through our projection, we can rely on autograd tools such as `PyTorch` (Paszke et al., 2019), except for the numerical optimization of the dual. Formally, the optimal dual η^* is a function of the LLM policy $\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$. To obtain a fully differentiable projection, we need the gradient $\frac{\partial \eta^*}{\partial \tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}$, which describes how the LLM output influences the optimal step size. We follow the OptNet framework (Amos et al., 2017) and differentiate the Karush-Kuhn-Tucker (KKT) conditions (Karush, 1939; Boyd et al., 2004) of the optimal dual solution via implicit differentiation (Dontchev et al., 2009) and matrix differential calculus (Magnus et al., 1989). This approach lets us compute the gradient in closed form instead of differentiating through the numerical optimization.

We visualize an exemplary gradient field of our projection in Figure 6.3. Inside the trust region, the gradients of the new policy remain unchanged. Outside of the trust region, the projection causes the gradients to point tangentially along the trust region boundary. This modification effectively removes gradients along the direction of the projection, which prevents updates that would move the policy further outside of the trust region along this direction. Combined with the regression loss in Equation 6.5, this behavior stabilizes training and prevents divergence from the old policy. Section D.1.3 provides detailed derivations and Section D.2 gives further details, including code snippets and a schematic of our projection's compute graph, including gradients.

Algorithm 4: Optimizing LLMs with *TROLL*

Input: LLM policy $\tilde{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$, Training data $\mathcal{D}$, KL Bound ϵ

for *step* $s{=}1\ldots$ **do**
  Sample batch of questions $\mathbf{q}{\sim}\mathcal{D}$
  Sample responses $\{\mathbf{o} \sim \tilde{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$ using the LLM policy
  Set reference policy $\pi_{\text{old}}{=}\tilde{\pi}_\theta$
  Sparsify and save corresponding logits $\pi_{\text{old}}(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$

  for *minibatch* b **do**
    Compute and sparsify current logits $\tilde{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$
    Estimate advantages A_t using any advantage estimation method

    for *response tokens* o *in parallel* **do**
      if *KL* $(\tilde{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t}) \,\|\, \pi_{old}(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})) \leq \epsilon$ **then**
        Set $\pi_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t}) = \tilde{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$ `// no projection needed`
      else
        Compute η^* by numerically optimizing Equation D.7
        `// project to` $\pi_{\text{old}}(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$
        Compute $\pi_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$ using Equation 6.4
      end
    end
    Update LLM parameters θ using Equation 6.5
  end
end

6.3.2. Sparse and Efficient Representations of Token Distributions

Naively implementing *TROLL* requires storing and projecting the full vocabulary distribution for each token. Using Qwen3's tokenizer (Yang et al., 2025b) as an example, this results in an overhead of 151 936 logits per token, which is prohibitively expensive. To address this issue, we sparsify both the distributions and the implementation of the projection. We greedily select the K tokens with the largest probability mass, sort them by their mass, and then only retain those needed to cover a cumulative mass of $1{-}\delta$. We additionally always keep the token actually selected by the LLM policy to ensure gradient information for this token. The top-K filtering both upper bounds the number of kept logits, acting as a fail-safe to prevent excessive memory usage for high-entropy predictions, and allows for efficient sorting of relevant tokens. Since pre-trained LLMs generally have very low perplexity (Kaplan et al., 2020; Hoffmann et al., 2022; Ruan et al., 2024), this thresholding allows us to maintain almost all of the probability mass of the logit distribution with very few average kept logits. Empirically, using $K{=}64$ and $\delta{=}10^{-5}$ usually allows us to keep 99.999% of probability mass with only $5{-}10$ average tokens for most of the tested model and task combinations. Finally, for the discarded tokens, we cannot assume a probability of truly 0 but have to use a small default mass $p_d{>}0$ to maintain well-behaved distributions.

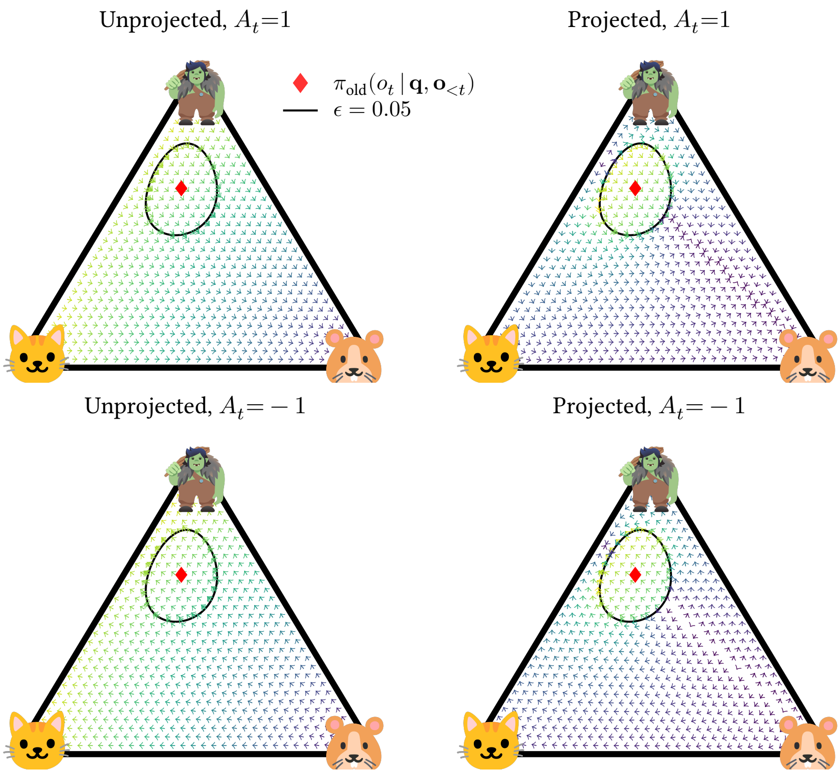


Figure 6.3.: Visualization of the gradient field for *TROLL*'s trust region projection for an exemplary 3-token distribution (cat, troll, hamster) after selecting the hamster token. (**Left**) Without *TROLL*'s projection, the gradient (arrows) either point toward the hamster or away from it, depending on the sign of the advantage A_t. The magnitude of the gradient scales with the distance to the hamster token. (**Right**) If the new policy is inside the KL bound of the old policy, its gradients remain unchanged. Outside of the bound, the policy is projected back onto the trust region, resulting in gradients that point tangentially along the trust region boundary. This behavior removes gradients orthogonal to the trust region, preventing divergence from the old policy along the direction of the projection.

After sparsification, we re-normalize the kept tokens with Equation D.15, taking into account the number of non-kept tokens and default mass. We perform the sparsification in chunks of the full generated sequences to prevent memory spikes.

While greedily keeping the highest-probability tokens is intuitively useful, we additionally show in Theorem D.1.1 in Section D.1.4 that it yields the best possible KL approximation under mild assumptions. Additionally, under moderate assumptions, the error introduced by sparsification is bound by

$$\mathrm{KL}\left(p \parallel q\right) \leq \gamma^{-1}\mathrm{KL}\left(p' \parallel q'\right) + \delta \log \frac{\delta}{q_{\min}}, \tag{6.6}$$

where p and q are arbitrary categorical distributions, p', q' the corresponding sparsified distributions, $q_{\min} \leq q(x_i)$ denotes a reference lower bound and $\gamma{\approx}1$ the renormalization constant. Theorem D.1.3 provides the proof and demonstrates that, for the hyperparameters used in Qwen3, the error incurred by enforcing the trust region on the sparsified distributions rather than on the full distributions is approximately two orders of magnitude smaller than the bound itself. The sparsification reduces memory and computation cost to the point where *TROLL* only incurs minimal overhead on modern LLMs, making it a practically viable alternative to PPO-like clipping. Further, this overhead is constant in model size, causing its relative cost to diminish for larger models. Figure 6.2 shows a schematic overview of *TROLL*'s training, and Section 6.6 provides additional analysis of the sparsification and projection behavior in practice.

6.4. Experiments

6.4.1. Datasets

We evaluate *TROLL* by finetuning LLMs for mathematical reasoning and code generation using an RLVR setup. For all math datasets, sequence-level binary rewards are computed by parsing the LLM output through a regular expression, matching against a ground truth answer. For code generation, we evaluate each answer using the provided test cases. The reward is then computed as the fraction of successful tests. The three mathematical reasoning tasks range from comparatively simple grade school problems to complex math olympiad tasks, and the code generation demands understanding of techniques such as dynamic programming, data structures and amortized analysis. We detail the datasets below. Section D.3.2 provides links to the individual datasets as well as an exemplary system pre-prompt.

DAPO-Math. DAPO-Math (Yu et al., 2025) consists of over 17 000 math questions and answers that are obtained from web scraping and manual annotation. Section D.5 provides an example question. We base our experiments on the version provided by Cui et al. (2025b), splitting it into `DAPO` Train and `DAPO` Eval versions. Concretely, we set aside 1 024 samples as an in-domain validation set (`DAPO` Eval), leaving 16 893 samples for `DAPO` Train.

Additionally, we follow the evaluation setup of Cui et al. (2025b) and use a suite of test datasets, which we call `Math`-Eval, comprised of MATH500 (Hendrycks et al., 2021), AMC, AIME2024 (Li et al., 2024b), AIME 2025, OMNI-MATH (Gao et al., 2025), Olympiad-Bench (He et al., 2024), and Minerva (Lewkowycz et al., 2022). As in previous work (Cui et al., 2025b), we report the mean of 32 rollouts for the comparatively small AIME2024, AIME2025, and AMC datasets to reduce evaluation variance, and consider a single response for the other datasets. We ensure all 3 datasets have the same system pre-prompt, which we provide in Listing D.4, and include correct and identical instructions for answer formatting.

GSM8K. `GSM8K` (Cobbe et al., 2021a) contains grade school math problems with final integer answers, consisting of 8.5k training and 1.3k test problems. We use the publicly available train and validation sets without further modifications.

Eurus-Math. Finally, we consider the publicly available train and validation sets of the Eurus-2-RL dataset (Cui et al., 2025a) , which is a subset of NuminaMath-CoT (Li et al., 2024b) and contains mathematical reasoning and code generation tasks. We filter for math questions, resulting in 455 261 train and 1 024 evaluation questions. We refer to the resulting dataset as `Eurus-Math` and use it as is.

Eurus-Code We use the same Eurus-2-RL dataset (Cui et al., 2025a) for code generation by filtering for code questions, resulting in 25 276 train and 1 024 evaluation questions. We call this subset `Eurus-Code`. Each includes multiple test cases of inputs and expected outputs after running the parsed python code within a sandbox. We use the PRIME reward manager of `verl` which evaluates up to the first ten test cases in the `SandboxFusion` sandbox. If the execution of a test case exceeds a timeout of ten seconds, the evaluation is terminated and zero reward is given.

The `Eurus-Code` dataset consists of tasks from APPS (Hendrycks et al., 2021), CodeContests (Li et al., 2022), TACO (Li et al., 2023) and Codeforces. For the evaluation, we compute the pass@1 scores for each of the four benchmarks and average the results. The validation split has 142, 377, 382 and 123 predefined questions for APPS, CodeContests, TACO, and Codeforces, respectively. The train data is split 13.7 %, 38.1 %, 37.9 % and 10.3 %, respectively, such that the evaluation overweights APPS and Codeforces. Empirically, the models seem to perform better on Codeforces but worse on TACO, such that the evaluation success rates are slightly higher.

6.4.2. Models

We experiment with Qwen3-{0.6B, 1.7B, 4B, 8B, 14B} (Yang et al., 2025a), which we use in thinking mode, and Qwen2.5-{0.5B,1.5B,3B,7B}-Instruct (Yang et al., 2025b). Furthermore, we include both the instruct and non-instruct versions of Llama-3.1-8B, Llama-3.2-3B (Grattafiori et al., 2024), and Apertus-8B (Hernández-Cano et al., 2025). Finally, we include Smol-LM3-3B Bakouch et al., 2025 and a version of Llama fine-tuned on FineMath (HuggingFaceTB, 2025). These models range from 500M to 14B parameters and cover different vocabulary sizes, tokenizers, model architectures, pre-training paradigms, and datasets, as well as initial math capabilities. Section D.3.1 provides download links for all pre-trained model checkpoints used in this work.

6.4.3. Methods

We focus on GRPO (Shao et al., 2024) due to its popularity and empirical success. We also include PPO (Schulman et al., 2017), which uses an explicit value model for Generalized Advantage Estimation (Schulman et al., 2015b), and three additional GRPO variants, namely Dr.GRPO (Liu et al., 2025), GSPO (Zheng et al., 2025), and REINFORCE++ (RF++) (Hu et al., 2025). All methods differ in their advantage estimation and in their normalization of Equation 6.1, yet they all rely on PPO-like clipping, which makes them amenable to using *TROLL*. We compare each method's original clipping-based version to using *TROLL* projections, denoted as (*Clip*) and (*TROLL*), respectively. Finally, we compare to BAPO (Xi et al., 2025) as an adaptive clipping method, and GPG (Chu et al., 2025) as a clipping-free baseline. For the latter, we compare to the vanilla variant (GPG) and also add the *TROLL* projection to its policy update (GPG (*TROLL*)).

6.4.4. Experiment Setup

We base our experiments on the `verl` repository, using default parameters and training recipes where applicable. We set the group size for the advantage normalization of all methods to 8. We use a token-level loss aggregation (Yu et al., 2025) for PPO and GRPO, and opt for method-specific loss aggregations for Dr.GRPO and GSPO. We evaluate the test datasets every 10 steps. To improve visibility, we use sliding windows of size 7 and 21 for the train and test evaluations, respectively, while showing the unsmoothed values in the background. Section D.3 provides additional details, including hyperparameters in Table D.3. Section D.4 shows all results.

Hardware. We train on clusters with Nvidia A100, H100, and H200 nodes, each equipped with 4 GPUs. For the Qwen3-14B, Qwen3-8B and Qwen2.5-7B-Instruct experiments in Section 6.5.1, we use H200s. For all other experiments, we use either H100 or A100 nodes, depending on model size. We train most experiments for up to 2 days, and extend some experiments on `DAPO` to up to 4 days to show algorithm convergence. We always train *Clip* and *TROLL* on identical hardware to ensure a fair comparison.

6.5. Results

6.5.1. Qwen experiments on `DAPO-Math`

Model Size. We evaluate models from the Qwen3 and Qwen2.5-Instruct families (Yang et al., 2025b; Yang et al., 2025a) on `DAPO` (Yu et al., 2025). The top of Figure 6.4 compares *TROLL* and the *Clip* objective for different Qwen3 model sizes optimized with GRPO (Shao et al., 2024). *TROLL* consistently improves training performance, causing more sample-efficient training and improved success rates at convergence for all models. These results directly translate to both evaluation sets, `DAPO`-Eval and `Math`-Eval. Interestingly, the 4B *TROLL* model almost matches the performance of the 14B *Clip* one. The right of Figure 6.1 further compares the runtime of both variants on Qwen3-14B, showing that *TROLL*'s projections do not incur a significant computational overhead. Finally, Section D.5 provides example sequences generated by Qwen3-14B on a `Math`-Eval problem.

Figure 6.5 extends the setup of Figure 6.4 to Qwen2.5-Instruct models. Similarly to the Qwen3 results, *TROLL* consistently improves over the *Clip* objective for each model size. We further find that, generally, most Qwen2.5 models slightly overfit on the training data, although this effect is less pronounced for *TROLL*.

https://github.com/volcengine/verl

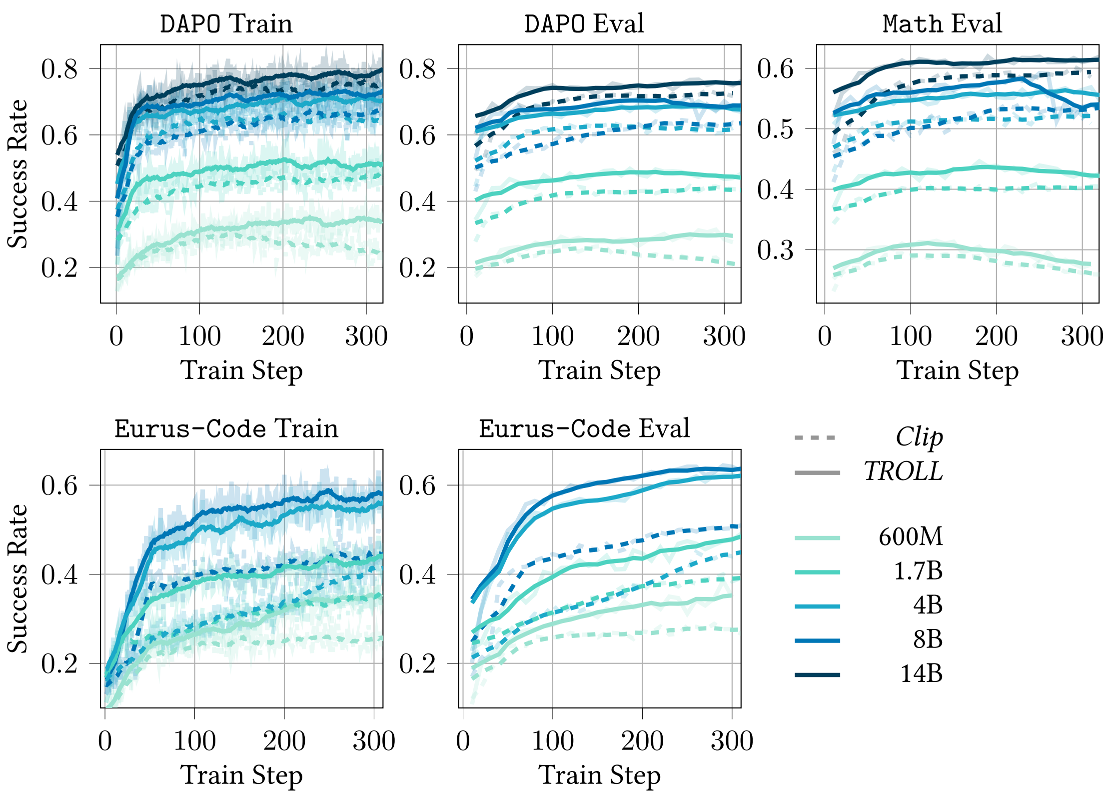


Figure 6.4.: Comparison of *TROLL* (full lines) and *Clip* (dashed lines) across GRPO-trained Qwen3 models with 600M to 14B parameters for `DAPO` (**top**) and `Eurus-Code` (**bottom**). Full-opacity lines mark smoothed results, while the background shows original values. *TROLL* consistently boosts training efficiency and final success rates on math-related questions and code generation tasks. These gains translate to different evaluation datasets.

		Qwen3-8B					Qwen2.5-7B-Instruct				
		GRPO	Dr.GRPO	PPO	GSPO	RF++	GRPO	Dr.GRPO	PPO	GSPO	RF++
`DAPO` Train	*Clip*	0.667	0.678	0.640	0.000	0.648	0.443	0.467	0.444	0.159	0.429
	TROLL	0.721	0.704	0.744	0.736	0.742	0.495	0.513	0.431	0.481	0.486
`DAPO` Eval.	*Clip*	0.640	0.653	0.602	0.000	0.626	0.323	0.331	0.324	0.093	0.323
	TROLL	0.691	0.674	0.715	0.706	0.728	0.389	0.389	0.353	0.390	0.380
`Math` Eval.	*Clip*	0.541	0.549	0.508	0.000	0.520	0.313	0.317	0.319	0.127	0.311
	TROLL	0.551	0.546	0.591	0.580	0.578	0.350	0.359	0.349	0.333	0.344

Table 6.1.: Final train and evaluation success rates on `DAPO` for Qwen3-8B and Qwen2.5-7B-Instruct methods for different advantage estimation methods for *TROLL* and *Clip*. The better approach is marked in blue. *TROLL* significantly improves over *Clip* in most cases, and is able to successfully train GSPO, where *Clip* causes divergence and little to no success rates on both models.

Advantage Estimation Methods. Table 6.1 compares final *TROLL* and *Clip* results for Qwen3-8B and Qwen2.5-7B-Instruct for GRPO, Dr.GRPO, PPO, GSPO, and RF++. Figure 6.6 and Figure 6.7 show detailed training and evaluation curves for the Qwen3 and Qwen2.5 setups, respectively. Finally, Table D.4 provides results for the individual `Math-Eval` datasets. We find that *TROLL* improves training success rates over *Clip* for both models

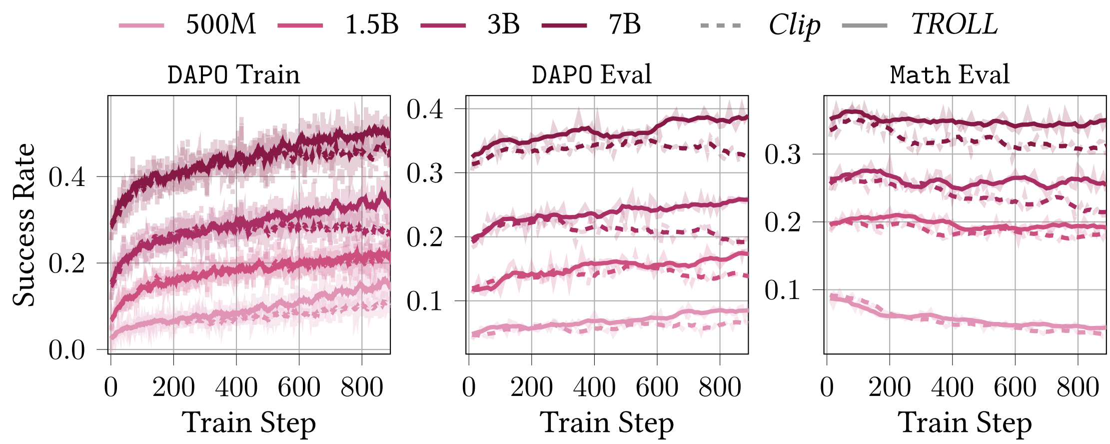


Figure 6.5.: Performance of *TROLL* and the *Clip* objective across Qwen2.5-Instruct models with 500M to 7B parameters trained with GRPO on DAPO. As in Figure 6.4, *TROLL* yields more sample-efficient training and higher rewards at convergence. These improvements extend both to evaluation on in-distribution questions and to generalization on out-of-distribution test datasets. Smoothed values are shown in full opacity, with original curves in the background.

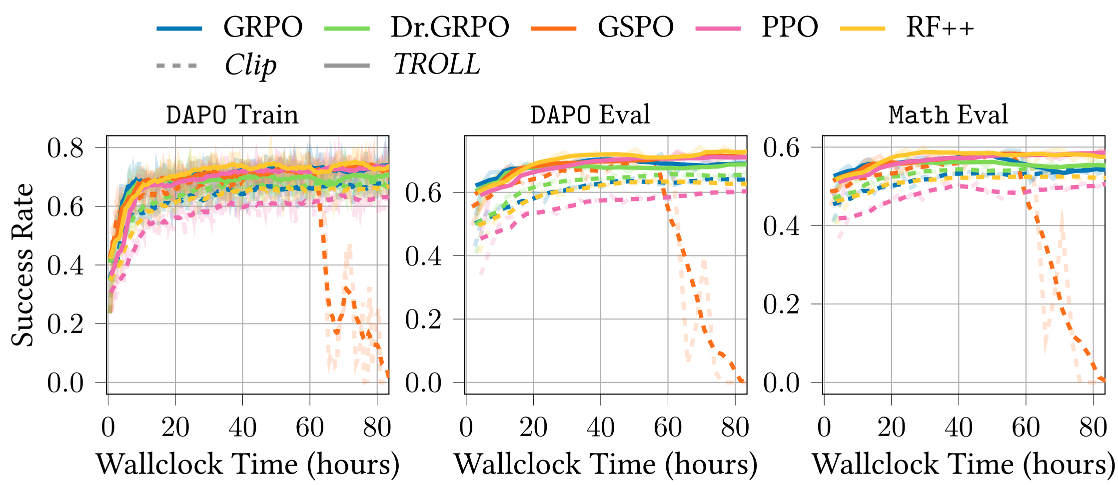


Figure 6.6.: *TROLL* and *Clip* success rates for Qwen3-8B trained with GRPO, Dr.GRPO, GSPO, PPO and RF++ on training data (**top**), in-domain evaluation (**bottom left**) and out-of-domain evaluation (**bottom right**). Smoothed values are shown in full opacity, with original curves in the background. *TROLL* improves over the *Clip* objective for all methods. For GSPO, *Clip* eventually diverges, leading to 0.00% success rate on all metrics, while *TROLL*'s optimization stays stable.

and across methods, to the point where Qwen3 GRPO and Dr.GRPO start to slightly overfit on the out-of-distribution `Math` evaluation. In these experiments, choosing *TROLL* over *Clip* is usually more beneficial than selecting any of the considered advantage estimation methods. Interestingly, *Clip* leads to unstable performance and eventual divergence for GSPO for both Qwen2.5 and Qwen3. In contrast, *TROLL*'s token-level trust region optimization remains stable and achieves similar success rates to the other advantage estimation methods.

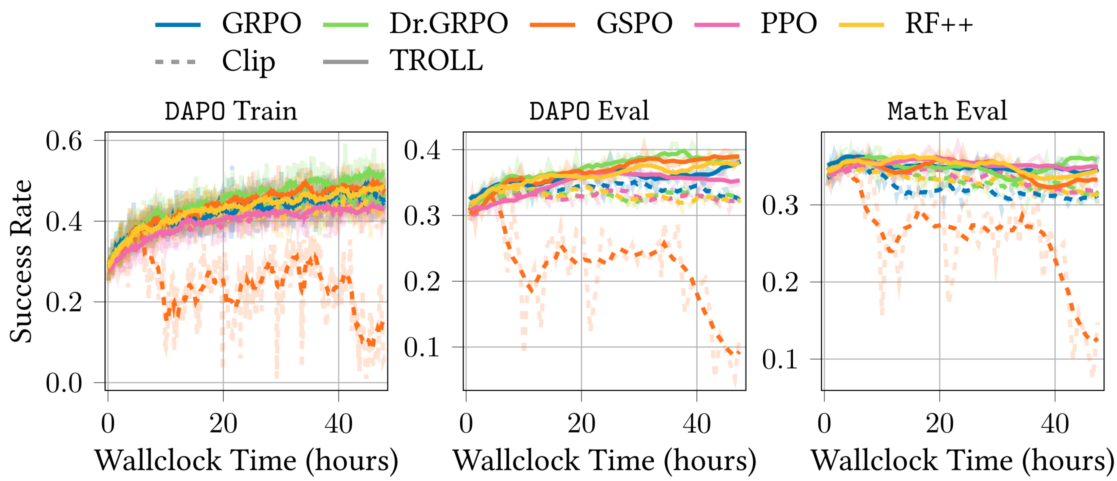


Figure 6.7.: *TROLL* and *Clip* success rates across Qwen2.5-7B-Instruct models trained with GRPO, Dr.GRPO, GSPO, PPO and RF++ on training data (**top**), in-domain evaluation (**bottom left**) and out-of-domain evaluation (**bottom right**). Smoothed values are shown in full opacity, with original curves in the background. *TROLL* improves over the *Clip* objective for all methods. For GSPO, *Clip* eventually diverges, while *TROLL*'s optimization stays stable.

6.5.2. Qwen3 on `Eurus-Code`

The bottom of Figure 6.4 shows that *TROLL* is directly applicable to code generation tasks, yielding substantial advantages over *Clip* for all model sizes on `Eurus-Code`. For instance, our Qwen3-1.7B (*TROLL*) improves over Qwen3-4B (*Clip*), and significantly outperforms the model of its own size. Specifically, we see improvements of $7-18$ percentage points, which translate to a $18-30\%$ relative gain, when using *TROLL* instead of *Clip*.

6.5.3. Additional Models and Datasets

Additional Model Families. Figure 6.8 shows various models of different families and sizes on `GSM8K`, again indicating a clear benefit for *TROLL* over the *Clip* objective. Here, models of the Llama family often need a significant number of training steps before *Clip* shows a positive training signal, while *TROLL* causes the models to start learning much faster. *TROLL* further enables stable and fast training for Apertus, while *Clip* fails in some setups. Section D.4.2 provides additional results on more models and `GSM8K`. We omit evaluations for `DAPO` with models other than SmolLM3-3B, as none matched the performance of Qwen3-1.7B in preliminary *Clip* experiments.

Additional Math Datasets. Considering other math datasets, the top rows on the left of Figure 6.8 show that *TROLL* is also beneficial on other datasets, as evaluated on `Eurus-Math` for Qwen3-8B and the simpler `GSM8K` for Qwen3-0.6B. Section D.4.3 provides detailed success rates for `Eurus-Math` in Figure D.5 and additional results on `GSM8K` for larger Qwen3 models in Figure D.4.

Model	Dataset	*Clip*	*TROLL*
Qwen3-0.6B	GSM8K	0.828	0.833
Qwen3-8B	Eurus-Math	0.561	0.579
SmolLM3-3B	GSM8K	0.915	0.925
SmolLM3-3B	DAPO	0.580	0.606
Llama3.2-3B	GSM8K	0.589	0.668
Llama3.2-3B Instr.	GSM8K	0.836	0.850
FineMath-Llama-3B	GSM8K	0.750	0.746
Llama3.1-8B	GSM8K	0.000	0.759
Llama3.1-8B Instr.	GSM8K	0.855	0.872
Apertus-8B	GSM8K	0.156	0.697
Apertus-8B Instr.	GSM8K	0.688	0.824

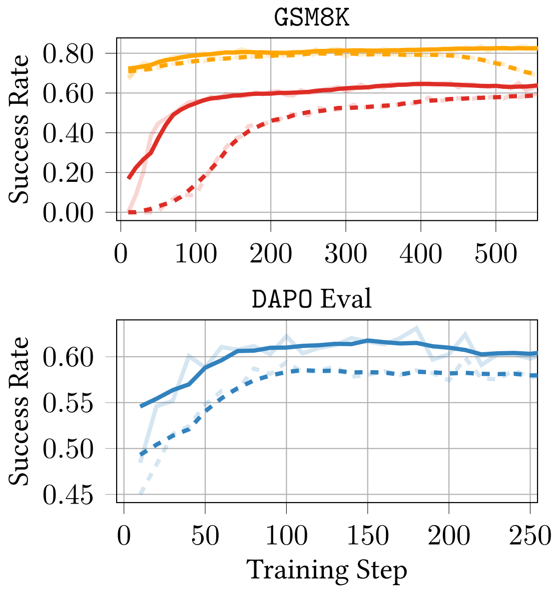


Figure 6.8.: (**Left**) Final evaluations for *TROLL* and *Clip* for different combinations of models and datasets trained with GRPO. The better approach between *TROLL* and *Clip* is marked in blue. (**Right**) Comparison of *TROLL* (full lines) and the *Clip* objective (dashed lines) for different models trained with GRPO. *TROLL* generally improves over *Clip*, and performs well across all considered datasets. In particular, *TROLL* leads to significantly faster learning for different Llama models, where *Clip* often takes significantly more iterations to obtain a positive training signal. *TROLL* also showcases more stable performance compared to *Clip* throughout training.

6.5.4. Additional Training Algorithms

Unlike PPO-style clipping, which relies on a ratio formulation, *TROLL* limits updates by enforcing trust regions directly on the LLM output. Thus, *TROLL* can in principle be applied to any LLM post-training algorithm. To verify this property, we evaluate *TROLL* on GPG (Chu et al., 2025), which is a clipping-free policy gradient method. Figure 6.9 finds that while clipping-free GPG initially learns well, it eventually diverges during training, likely due to unstable policy updates. Combining GPG with *TROLL* effectively resolves these stability issues, leading to performance on par with GRPO. These results indicate the efficacy of *TROLL*'s trust region projections in stabilizing training for non-clipping-based methods. The figure further evaluates BAPO (Xi et al., 2025), an adaptive clipping heuristic that can be used as an alternative to regular clipping. BAPO slightly improves over regular *Clip* on evaluation data, but still clearly underperforms *TROLL*.

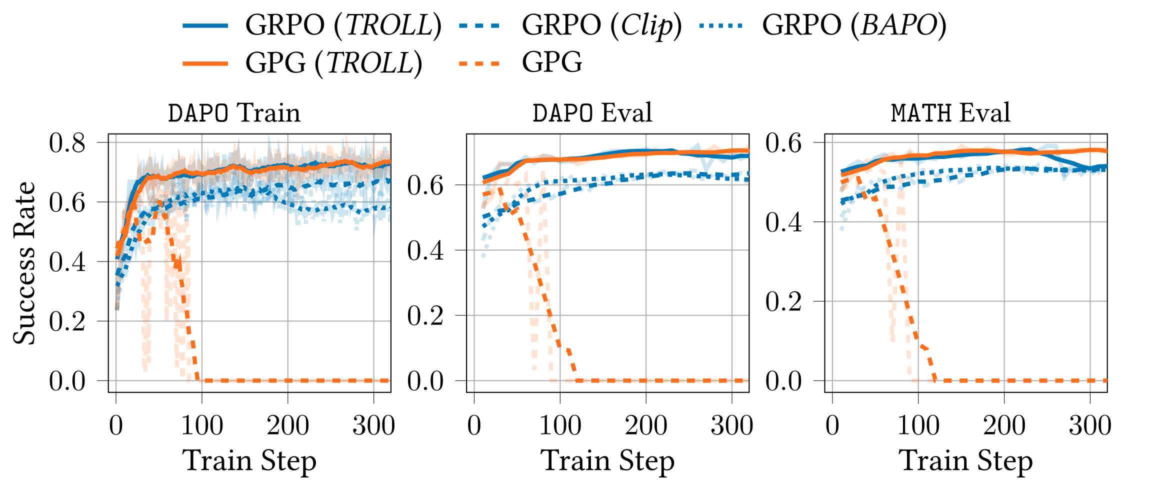


Figure 6.9.: Success rates for Qwen3-8B on DAPO for training data (**top**), in-domain evaluation (**bottom left**) and out-of-domain evaluation (**bottom right**). Smoothed values are shown in full opacity, with original curves in the background. *TROLL* performs well when combined with either GRPO or the clipping-free GPG baseline, while vanilla GPG eventually diverges. BAPO's adaptive clipping slightly improves over regular *Clip* on the evaluation datasets, but both underperform *TROLL* by a fair margin.

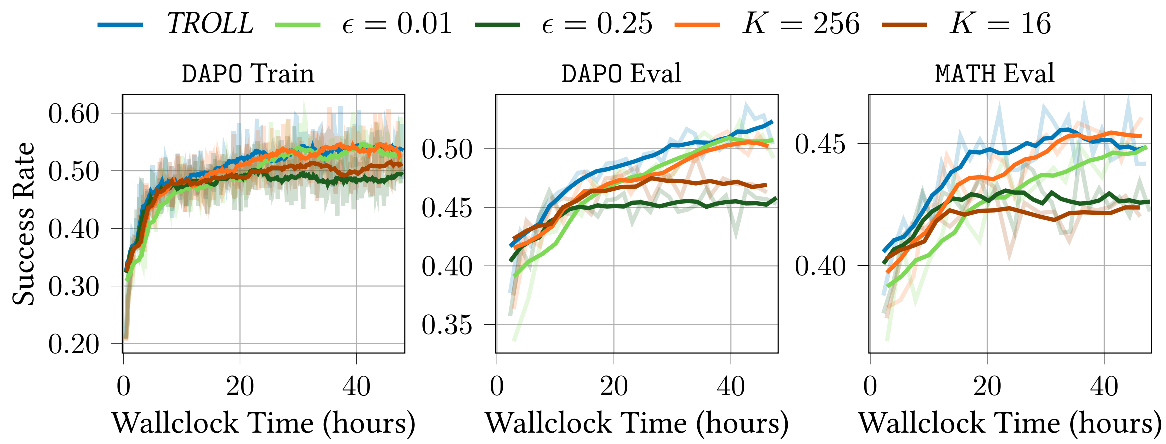


Figure 6.10.: Qwen3-1.7B trained with GRPO using the *TROLL* projection compared to different hyperparameter choices on DAPO training data (**top**), in-domain evaluation (**bottom left**) and out-of-domain evaluation (**bottom right**). Smoothed values are shown in full opacity, with original curves in the background. *TROLL* works well for reasonable KL bounds ϵ and top-K logit selections, but slightly degrades for too-large bounds and too few kept logits.

6.6. Analysis and Parameter Study

6.6.1. KL Bounds and Sparsity Threshold

We experiment with different KL bounds, testing ϵ=0.01 and ϵ=0.25 instead of the default ϵ=0.05. Additionally, we try different levels of sparsification. We switch the maximum number of kept tokens from K=64 to a lower K=16 and a higher K=256, adjusting the distribution mass threshold δ from $1e{-}5$ to $1e{-}4$ and $1e{-}6$ accordingly. Figure 6.10 shows that a lower KL bound ϵ for the projection leads to slower learning, but eventually reaches comparable performance. In contrast, a higher KL bound leads to worse performance, presumably because the policy moves too quickly during update steps. Reducing the number of kept tokens similarly leads to worse overall performance, which is likely caused by incorrect KL estimates of the original dense logit distributions. A higher amount of kept tokens does not yield any additional benefit. These results suggest that *TROLL* requires an accurate KL projection, while showing that there is a wide range of suitable hyperparameters for both the KL bound and the sparsification.

6.6.2. Generated Sequence Analysis

Figure 6.11 analyzes the behavior of sequences generated by different Qwen3 (**left**) and Qwen2.5 (**right**) model sizes trained on `DAPO` using GRPO. The top of the figure shows the fraction of selected tokens needed to satisfy *TROLL*'s sparsity mass threshold of $\delta{=}1e{-}5$ over training. There is a general trend that larger models require fewer selected tokens to satisfy the threshold, which is consistent with established LLM scaling laws (Kaplan et al., 2020). Here, as little as $5{-}10$ tokens are sufficient to capture most of the mass for the larger models. For the larger Qwen3 models, this trend appears less pronounced, likely because these models are to some extent saturating the `DAPO` benchmark.

In the middle of Figure 6.11, we analyze the fraction of clipped tokens for *Clip* compared to the fraction of projected tokens for *TROLL*. Both approaches affect roughly the same number of tokens, indicating that *TROLL*'s improvements are not merely caused by more restrained tokens. However, *TROLL*'s projection exhibits a lot more variance and tends to increase in later training stages, potentially suggesting a more active involvement in the learning process. In some cases *TROLL* projects a lot more aggressively, although this increase in projections does not cause a degradation in model performance.

Finally, the bottom of Figure 6.11 shows clear differences in response length dynamics over training. *TROLL* generally adapts the token length much faster than *Clip*, reducing the response length for Qwen3, while increasing it for Qwen2.5-Instruct. This difference originates in the different behavior of the pretrained models used to initialize learning, as the Qwen3 models tend to generate much longer responses, presumably due to their built-in thinking mode. After the RL fine-tuning with *TROLL*, the response lengths of both model families are more similar. In contrast, models trained with *Clip* show much slower shifts in response length. This quicker adjustment under *TROLL* aligns with the faster performance gains observed in both model families.

6.6.3. Batch Size Effects

TROLL's trust region projection promises more stable policy updates by constraining the difference between the new policy and the policy used to generate the data, in our case the responses. To evaluate this stability, we compare *TROLL* and *Clip* for different batch sizes.

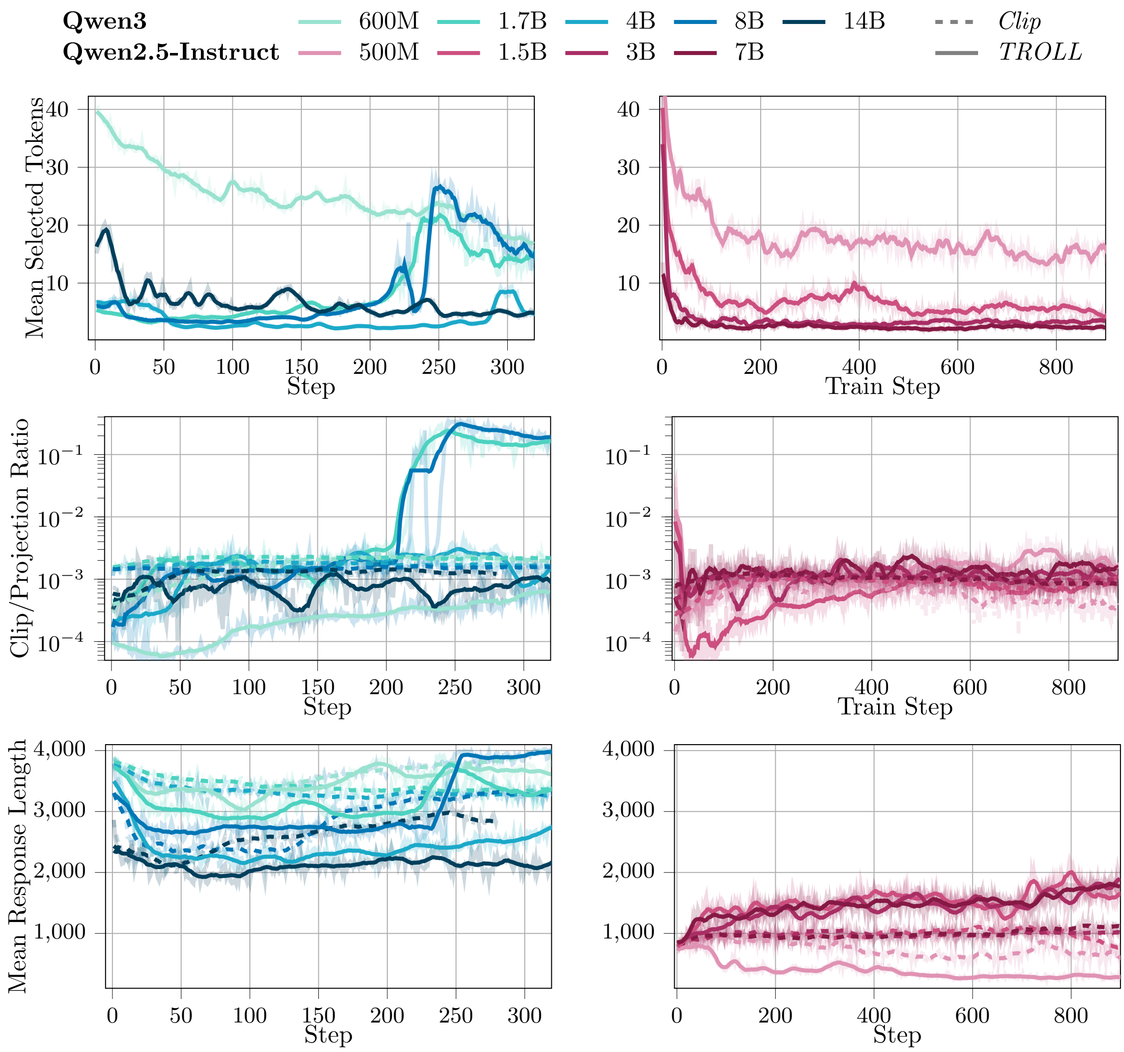


Figure 6.11.: Training dynamics of Qwen3 models on DAPO using GRPO. Smoothed values are shown in full opacity, with original curves in the background. Larger models need fewer tokens to meet the sparsity threshold of $\delta{=}10^{-5}$. While both approaches affect $\sim$0.1% of tokens most of the time, *TROLL* tends to increase projection later during training without harming performance. *TROLL* quickly adjusts response lengths while achieving higher success rates, whereas *Clip* is slower to alter the response length over time.

We use Qwen3-1.7B trained with GRPO on DAPO and leave all other parameters untouched. Doubling the batch size doubles the number of gradient steps between data collection, thus increasing the difference between the old policy $\pi_{\text{old}}(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$ and the current policy $\tilde{\pi}_\theta(o_t \,|\, \mathbf{q}, \mathbf{o}_{<t})$ over the course of the training iteration. Figure 6.12 shows that *TROLL* remains stable when increasing the batch size from our default of 256 to 2 048, indicating that *TROLL* can successfully train on older data due to its projection. In comparison, *Clip* shows a gradual degradation in performance for each doubling of the batch size, likely because the clipping does not address a potential divergence of the current policy to the policy that was used to generate the data.

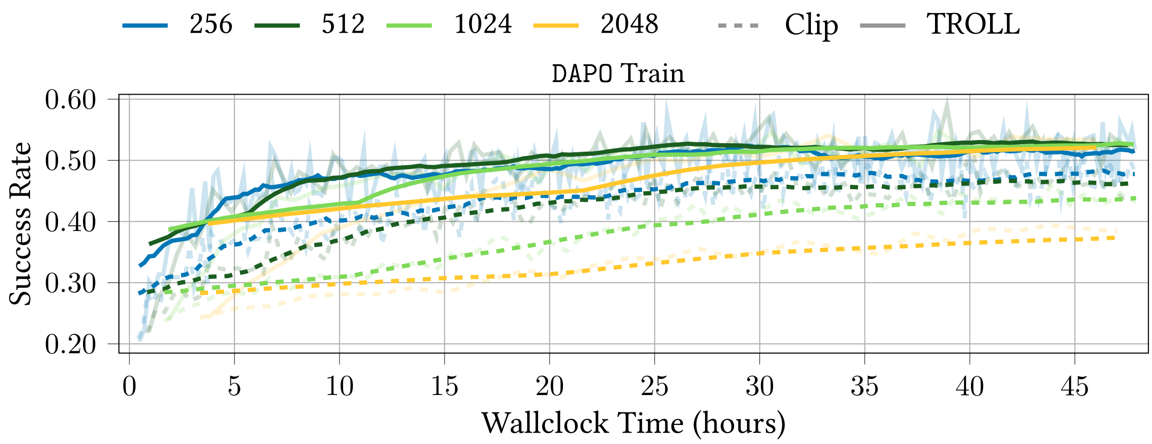


Figure 6.12.: *TROLL* and *Clip* training success rates for Qwen3-1.7B trained with GRPO on `DAPO` for different batch sizes. *TROLL* remains stable for larger batch sizes, while *Clip* gradually and consistently degrades when increasing the batch size.

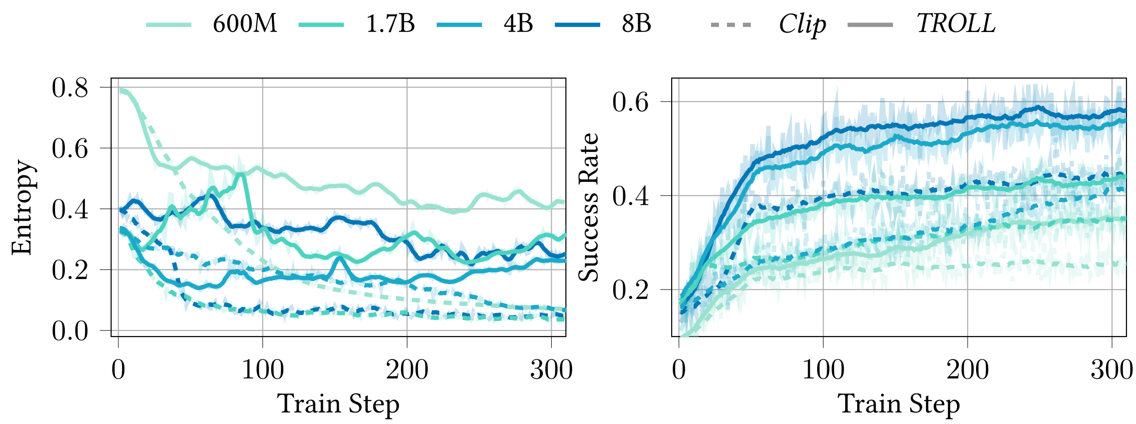


Figure 6.13.: *TROLL* and *Clip* entropy (**left**) and success rate (**right**) for different Qwen3 model sizes trained with GRPO on the `Eurus-Code` training data. Smoothed values are shown in full opacity, with original curves in the background. *TROLL* generally causes less decrease in token entropy while *Clip* shows a strong negative correlation between success rate and entropy. The quick improvement of Qwen3-8B *Clip* around step 40 coincides with a rapid drop in entropy.

6.6.4. Output Diversity and Entropy

Recent work has shown that the PPO-like *Clip* objective tends to exploit the LLM's existing knowledge by reducing each token distribution's entropy to increase the reward (Cui et al., 2025b). In contrast, Figure 6.13 and Figure 6.14 show a clear correlation between *TROLL*'s ability to preserve entropy and its improved performance on `Eurus-Code` and `Math-Eval`.

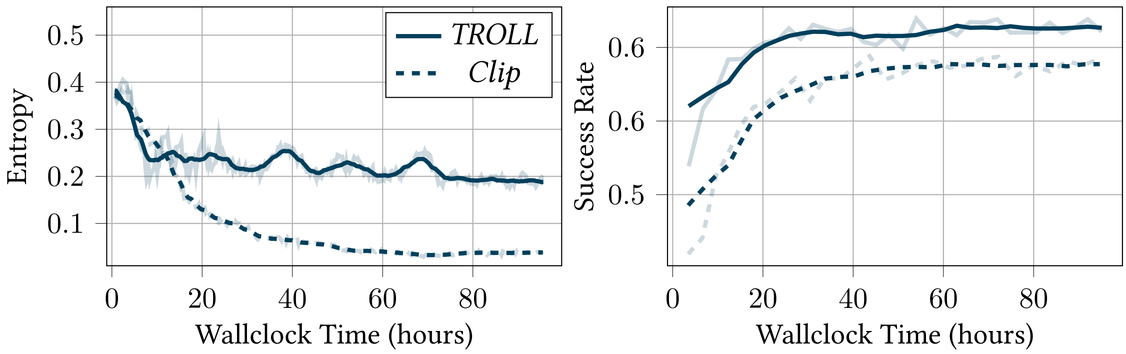


Figure 6.14.: *TROLL* and *Clip* training entropy (**left**) and evaluation success rate on `Math`-Eval (**right**) for Qwen3-14B trained with GRPO . *TROLL* maintains more entropy during training while showing higher success rates when compared to *Clip*.

Metric (Method)	Qwen3-0.6B	Qwen3-1.7B	Qwen3-4B
VRAM (*Clip*)	25.415 GiB	28.418 GiB	34.574 GiB
VRAM (*TROLL*)	27.868 GiB	30.663 GiB	36.837 GiB
VRAM (*TROLL* Chunked)	27.227 GiB	29.994 GiB	36.157 GiB
VRAM Delta (Chunked)	+1.812 GiB (+7.1%)	+1.576 GiB (+5.5%)	+1.583 GiB (+4.6%)
Runtime (*Clip*)	30.874s	43.372s	85.133s
Runtime (*TROLL*)	46.715s	49.053s	90.570s
Runtime (*TROLL* Chunked)	47.600s	50.629s	92.906s
Runtime Delta (Chunked)	+16.726s (+54.2%)	+7.257s (+16.7%)	+7.773s (+9.1%)

Table 6.2.: Max allocated VRAM per GPU and runtime of one iteration. The smallest 0.6B models do not fully saturate the GPU, so the Delta results differ from the larger models. The projection overhead is independent of the model size and already below ten percent for the small 4B model and slower A100 GPU. The advantage of the chunked sparsification depends on the micro batch size, so the benefit is larger for bigger GPUs.

6.6.5. Computational Overhead

We provide a simple numerical example to show the necessity of our sparsification scheme. For a small-scale training setup with a GRPO group size of 8, a sequence length of 256, and Qwen3's vocabulary of 151 936 tokens, storing dense float-32 probability distribution for a single prompt requires

$$256 \cdot 8 \cdot 151\,936 \cdot 4\text{B} \approx 1.16\ \text{GiB}$$

of memory. For each iteration, all methods need to store a rollout buffer of answers, in our case of size 256. In addition, the current mini-batch for the policy update needs to be stored. While this overhead can be reduced to a single answer with gradient accumulation, the rollout buffer still needs to store all outputs of the old policy. Together, this leads to a significant memory overhead, where storing only the dense distributions for a full batch of answers already requires roughly 296 GiB of memory. Since *TROLL* only needs to store sparse distributions, the memory requirements are drastically reduced. As seen in the top of Figure 6.11, sparsification requires an average of $5-10$ logits per token, which reduces the memory to less than 1MiB per prompt.

To measure the computational overhead of *TROLL*, including its sparsification, in terms of memory and runtime, we conduct a controlled experiment using `GSM8K` on four Nvidia A100-40GB GPUs. We use small Qwen3 models and our default GRPO setup. Measuring this overhead during standard training is infeasible because response lengths and the number of active tokens fluctuate and differ between models and algorithms. Furthermore, small models initially require significantly more logits to achieve the desired total probability mass due to higher initial perplexity. We therefore evaluate the models in their initial state, preventing weight updates via tight KL and clip ratio bounds. Note that simply setting the learning rate to zero is insufficient, as the LLM outputs must vary enough to trigger the trust region projection for a meaningful measurement. To ensure a fair comparison across model sizes, we clip all responses to 256 tokens. This length ensures that nearly all answers are truncated, resulting in an average length of >255 tokens across models, while still allowing the models to solve enough prompts to generate meaningful gradients.

As shown in Table 6.2, the sparsification and trust region projection results in a manageable experimental memory increase of only ≈ 6.3 GiB across four GPUs. *Chunked* refers to our default version of *TROLL*, where we normalize and sparsify the sequences in chunks of 1 024 tokens to avoid the dense single precision upcast of the entire mini batch. Further, both memory and computation time for *TROLL* scale primarily with vocabulary size. Since this remains constant within model families, the relative overhead of *TROLL* effectively diminishes as model size increases.

6.7. Conclusion

We introduce Trust Region Optimization for Large Language models (*TROLL*), a trust-region based policy gradient objective that acts as a drop-in replacement for the popular PPO-clip. *TROLL* is based on a novel, principled, and fully differentiable trust-region projection for discrete distributions. This projection compares two distributions, in our case, the token logit distributions of an old policy used to collect sequences, and a new policy that performs policy gradient updates on these sequences. Since these distributions are prohibitively large for modern vocabulary sizes, we extend the projection to sparse distributions. Here, we only keep a small subset of logits that represent the most likely token predictions, allowing us to realize both data collection and the projection objective using fully sparse operations. We experimentally validate *TROLL* across various model families, model sizes, advantage estimation methods, and math and code generation datasets. *TROLL* significantly and consistently outperforms the PPO-clip objective in terms of sample efficiency and final reward across setups, while only requiring a small overhead that does not scale with model size.

6.7.1. Limitations and Future Work

We currently evaluate our method on dense models with up to 14B parameters. In future work, we want to scale *TROLL* to larger models and Mixture-of-Experts architectures. Similarly, it would be interesting to extend *TROLL* to other modalities and tasks, using, for example, vision-language models, where the logit distributions and their projections may behave differently from pure language.

6.7.2. Broader Impact

TROLL improves the efficiency of LLM fine-tuning by enabling scalable trust-region optimization. While our experiments focus on mathematical reasoning, the method is broadly applicable to other domains. As with any advance in LLM training, this carries both potential benefits and risks, depending on the context of deployment. We believe that managing and shaping the societal impacts of increasingly powerful LLMs should not be left to individual researchers, organizations, or companies alone, but rather that they must be carefully governed and regulated by sovereign governments and strong democratic institutions.

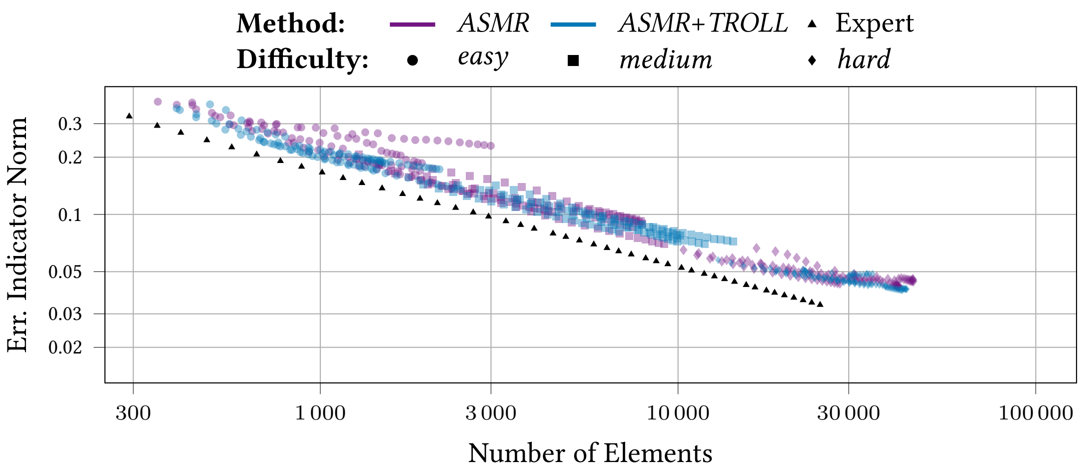


Figure 6.15.: Log-log plot of error indicator norm versus number of mesh elements (lower left is better) for *ASMR*, *ASMR+TROLL* and the expert across `L-Shape` (*easy*, *medium*, *hard*). Each marker shows the mean over the test set for a given seed. *ASMR+TROLL* modestly improves over *ASMR* alone. On average, it achieves a lower error for a given element budget, and more closely matches expert performance.

6.8. Combining *TROLL* with *ASMR*

After assessing *TROLL*'s efficacy for LLMs in the previous sections, we now briefly investigate its compatibility with Adaptive Swarm Mesh Refinement (*ASMR*) as a RL-based Adaptive Mesh Refinement (AMR) framework. Since *ASMR* considers binary actions per mesh element, we omit the logit sparsification of Section 6.3.2. We further note that the binary action space allows recovering the unselected action's probability directly from the selected action's probability, i.e., $\pi_\theta(a_{j,t} = 0|\mathcal{G}_{M^t}) = 1 - \pi_\theta(a_{j,t} = 1|\mathcal{G}_{M^t})$ for element j of mesh M^t, allowing the use of *TROLL* without any modification to the replay buffer.

We follow the *ASMR* setup from Section 5.4.4 comparing *ASMR* with and without *TROLL* on the `L-Shape` (*easy*, *medium*, *hard*) mesh refinement settings. We leave all *ASMR* hyperparameters unchanged, only slightly adjusting the element penalty term for consistency and replacing PPO's *Clip* objective with *TROLL*' objective as described in Equation 6.5. We keep the trust region bound of $\epsilon = 0.05$, but use a regression alpha of $\alpha = 10$ for *TROLL* to adjust to the different action space of *ASMR* and to ensure stable training. Figure 6.15 shows the resulting element-error Pareto plot. The figure overlays the results of Figure 5.5 with those of *ASMR+TROLL*, omitting *AMBER* for clarity. We observe that *ASMR+TROLL* slightly but consistently outperforms *ASMR* alone, achieving lower error for a given element budget, and thus more closely matching the performance of the expert heuristic. This result indicates that *TROLL* is directly compatible with RL-based AMR frameworks, highlighting its versatility and potential application in engineering domains.

7. Conclusion

This thesis addresses the challenge of grounding agent perception and optimization in the intrinsic, geometric structure of a given task. As machine learning has shifted from isolated, static domains toward agents operating in complex environments and physical domains, standard image-based or vector-based representations and heuristic optimization schemes prove insufficient. Conventional approaches often fail to account for the symmetries and topological constraints of the physical world, causing them to require excessive amounts of data to generalize beyond simple training scenarios. Even then, the absence of structural awareness means that even with abundant data, the resulting models often remain brittle and struggle to identify the underlying invariances required for widespread adaptation. Simultaneously, training an agent tends to rely on local approximations of the target objective, but standard optimization algorithms only superficially account for the intrinsic geometry of the policy space during updates, which can lead to unstable learning, especially in high-dimensional action spaces. We integrate principles from Geometric Deep Learning and Information Geometry into Reinforcement Learning (RL) and Imitation Learning (IL) frameworks, and demonstrate that these structural priors are paramount for developing robust, efficient, and generalizable learning agents.

To facilitate and guide our investigation into geometry-aware agents, we formulated three research questions in Chapter 1:

- How can we utilize geometric representations to capture versatile human expert behavior in robotic manipulation? (Chapter 3)
- How can we enable learning agents to leverage intrinsic geometric features to represent, reason over, and optimize the discretization of irregular physical domains? (Chapter 4 and Chapter 5)
- How can we employ information-geometric constraints to stabilize policy optimization in high-dimensional action spaces? (Chapter 6)

Guided by these questions, we have shown that geometric priors allow agents to organize otherwise unstructured problem spaces. We introduced geometric behavioral descriptors that resolve the ambiguity of multi-modal human demonstrations by focusing on relative relations rather than absolute coordinates. We developed architectures that treat physical meshes as dynamic graphs, enabling adaptive simulations that scale to complex, irregular topologies. Finally, we replaced crude optimization heuristics with principled information-geometric projections to ensure training stability even in extremely high-dimensional action spaces. Collectively, these contributions provide a framework for agents that are fundamentally aligned with the geometric structure of the problems they solve.

7.1. Summary of Contributions

The research presented in this thesis is comprised of several distinct works, with each chapter addressing a specific facet of geometry in learning agents. The following paragraphs summarize the key contributions of each chapter and the original work presented therein.

Chapter 3 addresses the first research question by considering the problem of capturing versatile human behavior in IL for robotic manipulation. Standard IL often struggles with the multi-modality of human demonstrations. When an expert provides multiple ways to solve a task, traditional algorithms tend to average over these distinct strategies. The result is a "blending" of trajectories that often fails to achieve the goal and may even lead to unsafe or dangerous behavior in real-world applications. We introduce Versatile Imitation from Geometrically Observed Representations (*VIGOR*), which utilizes geometric descriptors to encode task states as relative relations. By transforming the problem into a space where similar performances appear similar regardless of the absolute Euclidean coordinates of the execution, the agent can generalize to novel configurations with minimal data. We combine this representation with a mode-seeking Information-Projection objective, which allows the agent to capture distinct expert strategies. The underlying optimization relies on trust-region-constrained policy search, constraining each mixture component's update based on the policy's distribution manifold. Experiments on mixture models show that this approach effectively captures the versatility of a human expert, while enabling robust generalization to new contexts from few demonstrations.

Chapter 4 contributes to the second research question by investigating how RL can be used to develop Adaptive Mesh Refinement (AMR) strategies. Accurate simulation of complex systems is essential for engineering but often carries a prohibitive computational cost. Adaptive meshing reduces this cost by dynamically placing mesh elements on a given domain. However, classical approaches rely on hand-tuned, often task-specific heuristics or expensive error estimators that do not scale. Our approach, Adaptive Swarm Mesh Refinement (*ASMR*), treats the mesh as a geometric graph and the refinement process as a collaborative multi-agent learning problem. In this framework, each mesh element acts as an autonomous agent within a homogeneous swarm. To account for the evolving geometry of the mesh, we developed a learning scheme that accommodates changing agent populations over the course of multiple refinement steps. By providing local, element-wise rewards depending on the local simulation quality, and mapping these rewards to the responsible agents over time, the system learns to focus computational resources on regions of high simulation error. Our results demonstrate that this swarm-based approach matches the performance of expensive oracle strategies while producing meshes that simulate up to two orders of magnitude faster than uniform refinement.

Chapter 5 complements the previous chapter by exploring IL for iterative Adaptive Mesh Generation (AMG). We introduce Adaptive Meshing By Expert Reconstruction (*AMBER*), a framework that learns to predict a target resolution field directly from expert examples. Instead of viewing the mesh as a collaborative swarm of agents, *AMBER* treats it as a single entity that must learn to locally adapt its discretization. Starting on some coarse mesh, the agent iteratively predicts a target resolution for each element and generates a new mesh that adheres to this target resolution. This process is repeated for several steps, causing the mesh to converge to an expert-like design. To this end, *AMBER* leverages a hierarchical GNN to simultaneously reason about the mesh on local and global scales. Equipped with this architecture, *AMBER* exploits the geometry of the task through two levels of adaptivity. First, the hierarchical architecture naturally adapts to the irregular and spatially varying topology of the physical domain, allowing for consistent communication across varying mesh densities. Second, the iterative framework allows the agent to dynamically refine its own discretization, where the policy's previous decisions dictate the resolution and structure of the intermediate meshes it subsequently reasons over. To ensure stability during this iterative process, we train on a replay buffer of agent-generated meshes that are automatically labeled through projections of the expert mesh. Evaluations on complex physical problems and various highly irregular domains demonstrate that *AMBER* generalizes to unseen geometries and significantly outperforms recent learning-based methods in generating high-quality meshes.

Chapter 6 concludes our investigation by addressing the third question. It examines how information-geometric constraints can be used to stabilize the optimization of complex learning agents. Instead of focusing on the geometry of the external environment, this chapter is about the internal manifold of the agent's policy distribution. In high-dimensional settings like LLM fine-tuning, standard optimization often relies on clipping-based heuristics. Clipping acts as a simplistic approximation of a trust region, but truncate gradients and offer no formal guarantees of stability, leading to less efficient and more erratic learning behavior. We propose Trust Region Optimization for Large Language models (*TROLL*), a method that replaces these unprincipled heuristics with a discrete and differentiable trust-region projection that uses the geometry of the policy manifold as an inductive bias for stable updates. By applying principled constraints to a sparse subset of the most critical suggested tokens, we ensure that policy updates remain within a theoretically sound proximity of the data-collection policy without sacrificing computational efficiency. We show that, on language-centric tasks like mathematical reasoning and code generation, *TROLL* consistently outperforms standard clipping in terms of training speed, stability, and final success rates, in some cases by more than 10 percentage points. Combining *TROLL* with *ASMR* demonstrates that these benefits largely generalize to AMR for physical simulation.

These contributions individually and collectively demonstrate the benefits of integrating geometric priors into the representation and optimization of learning agents. By grounding machine learning in the intrinsic structure of the task, this thesis establishes a principled foundation for building more robust, efficient, and scalable autonomous systems.

7.2. Outlook and Future Work

Geometry and the structure that it brings to learning agents represent a vast and vastly powerful toolkit. Be it through geometric priors, symmetry-aware architectures acting on irregular domains, or information-geometric optimization, equipping agents with the ability to reason over the geometry of their problem space is crucial for developing robust and generalizable behavior. In this sense, the advancements proposed in this thesis are a small step toward the full potential of geometry-aware agents, and there remain many promising directions for future research in this area. This potential is perhaps most evident in the recent shift toward generalist foundation models (Bommasani, 2021; Firoozi et al., 2025; Awais et al., 2025; Zhou et al., 2025). The efficacy of these models relies equally on the capacity of their billion-parameter backbones, and the availability of sufficiently large and diverse datasets to train them on. However, it remains to be seen if performance will bottleneck once high-quality, human-generated data sources are exhausted (Shumailov et al., 2024; Villalobos et al., 2024). As we approach these limits, we believe that more sophisticated, geometry-aware input representations, architectures and optimization schemes will be crucial for the next generation of models to continue improving in performance and generalization.

Particularly in the field of robot learning, recent years have seen a surge in increasingly capable foundation models (Black et al., 2024; Kim et al., 2024; Reuss et al., 2025; Intelligence et al., 2025). Foundation models in robot learning often rely on score-based diffusion (Song et al., 2021; Scheikl et al., 2024; Chi et al., 2025) or flow matching (Lipman et al., 2023; Black et al., 2024) to learn complex, multi-modal expert behavior. These generative objectives are significantly more expressive and capable than the mode-seeking Information-Projection used in Versatile Imitation from Geometrically Observed Representations (*VIGOR*), and have largely superseded the broader class of discriminator-based objectives (Goodfellow et al., 2014) to which *VIGOR* belongs. However, they come at a higher inference cost and implicitly rely on model capacity to prevent mode averaging and collapse. Further, many robotic foundation models utilize simple vision-based representations that do not explicitly account for the underlying geometry of the task, which may limit their generalization to novel scenarios and configurations. Exploring how to integrate insights from *VIGOR* into these modern architectures could bridge the gap between high-capacity generative modeling and both grounded geometric representations and robust, mode-seeking objectives. Similarly, Trust Region Optimization for Large Language models (*TROLL*)'s information-geometric trust region projection could be applied to the training of these models, where the high-dimensional action space and the need for stable optimization are particularly pronounced. These integrations have the potential to improve data efficiency and robustness while also reducing inference cost.

Learned surrogate models for physical simulation have seen a significant surge in interest, shifting from task-specific solvers toward more expansive, general-purpose architectures. While current efforts to develop physics foundation models often rely on specific tokenization schemes coupled with transformers (Alkin et al., 2024; Alkin et al., 2025; Wiesner et al., 2025; Nguyen et al., 2025), mesh-based representations remain the standard for many specialized domains that require geometric fidelity. This trend is evident in recent advancements across solid mechanics (Würth et al., 2025; Dahlinger et al., 2025), biomedical surface and volume modeling (Suk et al., 2024), and weather forecasting (Lam et al., 2023; Price et al., 2025). In such mesh-based learned simulators, the quality of the input mesh plays an important role in scaling to complex, real-world problems by focusing computational resources on important regions of the simulated domain. Consequently, integrating agent-based adaptive meshing methods such as Adaptive Swarm Mesh Refinement (*ASMR*) or Adaptive Meshing By Expert Reconstruction (*AMBER*) into these models could significantly enhance their performance and applicability. Said integration can be realized either as a decoupled pre-processing step, where the meshing method provides an optimized domain discretization for the surrogate, or with a more tightly coupled approach where both models are trained jointly in an end-to-end manner.

In the context of Large Language Models (LLMs), foundation models already see widespread use, to the point where such models impact entire industries and transform day-to-day workflows. Here, model alignment (Rafailov et al., 2023) and post-training (Shao et al., 2024) are essential to ensuring that pre-trained models become safe, reliable, and effective for real-world applications. *TROLL* provides an information-theoretic objective for such training processes through its trust region projections, significantly improving over current clipping-based alternatives. This increased efficiency can directly reduce the computational cost and carbon footprint of training and fine-tuning LLMs, while improving the quality of the resulting model. Further, *TROLL* is only the first step toward more sophisticated trust-region methods for LLM post-training, with future directions potentially including explicit entropy control, an adaptive trust region bound, and off-policy extensions that re-use generated sequences during optimization.

To summarize, recent research has seen an increased shift toward generalist foundation models across various domains. We believe that the integration of geometry-aware representations, architectures, and optimization schemes will be pivotal for future foundation models, potentially leading to improvements across data-efficiency, robustness, performance, and generalization.

Bibliography

Abbeel, P. and A. Y. Ng (2004). "Apprenticeship learning via inverse reinforcement learning". In: *Proceedings of the Twenty-First International Conference on Machine Learning (ICML)*. ICML '04, p. 1. DOI: 10.1145/1015330.1015430. URL: https://doi.org/10.1145/1015330.1015430.

Abdolmaleki, A., R. Lioutikov, J. R. Peters, N. Lau, L. Pualo Reis, and G. Neumann (2015). "Model-Based Relative Entropy Stochastic Search". In: *Advances in Neural Information Processing Systems (NIPS)*.

Abdolmaleki, A., J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller (2018). "Maximum a Posteriori Policy Optimisation". en. In: *arXiv:1806.06920 [cs, math, stat]*. (Visited on 2019-03-11).

Abdullah, S., N. Al-Asady, A. Ariffin, and M. Rahman (2008). "A review on finite element analysis approaches in durability assessment of automotive components". In: *Journal of Applied Sciences* 8.12, pp. 2192–2201.

Agarwal, R., M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare (2021). "Deep Reinforcement Learning at the Edge of the Statistical Precipice". In: *Advances in Neural Information Processing Systems (NeurIPS)* 34. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, pp. 29304–29320. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/f514cec81cb148559cf475e7426eed5e-Paper.pdf.

Akkaya, I., M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. (2019). "Solving rubik's cube with a robot hand". In: *arXiv preprint arXiv:1910.07113*.

Akrour, R., A. Abdolmaleki, H. Abdulsamad, J. Peters, and G. Neumann (2018). "Model-Free Trajectory-based Policy Optimization with Monotonic Improvement". en. In.

Akrour, R., J. Pajarinen, J. Peters, and G. Neumann (2019). "Projections for Approximate Policy Iteration Algorithms". In: *Proceedings of Machine Learning Research*, pp. 181–190.

Alkin, B., M. Bleeker, R. Kurle, T. Kronlachner, R. Sonnleitner, M. Dorfer, and J. Brandstetter (2025). "AB-UPT: Scaling Neural CFD Surrogates for High-Fidelity Automotive Aerodynamics Simulations via Anchored-Branched Universal Physics Transformers". In: *arXiv preprint arXiv:2502.09692*.

Alkin, B., A. Fürst, S. Schmid, L. Gruber, M. Holzleitner, and J. Brandstetter (2024). "Universal physics transformers: A framework for efficiently scaling neural operators". In: *Advances in Neural Information Processing Systems* 37, pp. 25152–25194.

Allen, K. R., T. L. Guevara, Y. Rubanova, K. Stachenfeld, A. Sanchez-Gonzalez, P. Battaglia, and T. Pfaff (2022). "Graph network simulators can learn discontinuous, rigid contact dynamics". In: *Conference on Robot Learning (CoRL)*.

Allen, K. R., Y. Rubanova, T. Lopez-Guevara, W. F. Whitney, A. Sanchez-Gonzalez, P. Battaglia, and T. Pfaff (2023). "Learning rigid dynamics with face interaction graph networks". In: *International Conference on Learning Representations (ICLR).*

Amari, S.-i. (2016). *Information geometry and its applications.* Vol. 194. Springer.

Amos, B. and J. Z. Kolter (2017). "OptNet: Differentiable Optimization as a Layer in Neural Networks". In: *arXiv:1703.00443 [cs, math, stat].* (Visited on 2018-11-13).

Anderson, R., J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cerveny, V. Dobrev, Y. Dudouit, A. Fisher, T. Kolev, et al. (2021). "MFEM: A modular finite element methods library". In: *Computers & Mathematics with Applications* 81, pp. 42–74.

Andrychowicz, M., A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem (2021). "What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study". In: *International Conference on Learning Representations (ICLR).* URL: `https://openreview.net/forum?id=nIAxjsniDzg`.

Antman, S. S. (2005). "Problems in nonlinear elasticity". In: *Nonlinear Problems of Elasticity*, pp. 513–584.

Arenz, O., G. Neumann, and M. Zhong (2018). "Efficient Gradient-Free Variational Inference using Policy Search". In: *International Conference on Machine Learning (ICML).*

Arenz, O., M. Zhong, and G. Neumann (2020). "Trust-Region Variational Inference with Gaussian Mixture Models." In: *J. Mach. Learn. Res.* 21, pp. 163–1.

Argall, B. D., S. Chernova, M. Veloso, and B. Browning (2009). "A survey of robot learning from demonstration". In: *Robotics and Autonomous Systems* 57.5, pp. 469–483.

Arnold, D. N., A. Mukherjee, and L. Pouly (2000). "Locally adapted tetrahedral meshes using bisection". In: *SIAM Journal on Scientific Computing* 22.2, pp. 431–448.

Awais, M., M. Naseer, S. Khan, R. M. Anwer, H. Cholakkal, M. Shah, M.-H. Yang, and F. S. Khan (2025). "Foundation models defining a new era in vision: a survey and outlook". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

Ba, L. J., J. R. Kiros, and G. E. Hinton (2016). "Layer Normalization". In: *CoRR* abs/1607.06450, p. 21. arXiv: `1607.06450`.

Bain, M. and C. Sammut (1995). "A Framework for Behavioural Cloning." In: *Machine Intelligence 15*, pp. 103–129.

Baker, B., I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch (2020). "Emergent Tool Use From Multi-Agent Autocurricula". In: *International Conference on Learning Representations.* URL: `https://openreview.net/forum?id=SkxpxJBKwS`.

Baker, T. J. (1997). "Mesh adaptation strategies for problems in fluid dynamics". In: *Finite Elements in Analysis and Design* 25.3-4, pp. 243–273.

– (2005). "Mesh generation: Art or science?" In: *Progress in aerospace sciences* 41.1, pp. 29–63.

Bakouch, E., L. Ben Allal, A. Lozhkov, N. Tazi, L. Tunstall, C. M. Patiño, E. Beeching, A. Roucher, A. J. Reedi, Q. Gallouédec, K. Rasul, N. Habib, et al. (2025). *SmolLM3: smol, multilingual, long-context reasoner.* URL: `https://huggingface.co/blog/smollm3`.

Bangerth, W., C. Burstedde, T. Heister, and M. Kronbichler (2012). "Algorithms and data structures for massively parallel generic adaptive finite element codes". In: *ACM Transactions on Mathematical Software (TOMS)* 38.2, pp. 1–28.

Bangerth, W. and R. Rannacher (2013). *Adaptive Finite Element Methods for Differential Equations.* Birkhäuser.

Baum, M., D. Anders, and T. Reinicke (2023). "Approaches for numerical modeling and simulation of the filling phase in injection molding: A Review". In: *Polymers* 15.21, p. 4220.

Becker, P., O. Arenz, and G. Neumann (2020). "Expected Information Maximization: Using the I-Projection for Mixture Density Estimation". In: *International Conference on Learning Representations (ICLR).*

Becker, P., N. Freymuth, S. Thilges, F. Otto, and G. Neumann (2026). "TROLL: Trust Regions improve Reinforcement Learning for Large Language Models". In: *International Conference on Learning Representations (ICLR).*

Becker, R., G. Gantner, M. Innerberger, and D. Praetorius (2023). "Goal-oriented adaptive finite element methods with optimal computational complexity". In: *Numerische Mathematik* 153.1, pp. 111–140.

Bellman, R. (1957). "A Markovian decision process". In: *Journal of mathematics and mechanics* 6.5, pp. 679–684.

Bentley, J. L. (1975). "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM* 18.9, pp. 509–517.

Berger, M. J. and P. Colella (1989). "Local adaptive mesh refinement for shock hydrodynamics". In: *Journal of computational Physics* 82.1, pp. 64–84.

Berner, C., G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. (2019). "Dota 2 with large scale deep reinforcement learning". In: *arXiv preprint arXiv:1912.06680.*

Binev, P., W. Dahmen, and R. DeVore (2004). "Adaptive finite element methods with convergence rates". In: *Numerische Mathematik* 97, pp. 219–268.

Bishop, C. M. (1994). *Mixture density networks.* Aston University.

Bjorck, N., C. P. Gomes, and K. Q. Weinberger (2021). "Towards deeper deep reinforcement learning with spectral normalization". In: *Advances in Neural Information Processing Systems (NeurIPS)* 34, pp. 8242–8255.

Black, K., N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. (2024). "π_0: A Vision-Language-Action Flow Model for General Robot Control". In: *arXiv preprint arXiv:2410.24164.*

Blei, D. M., A. Kucukelbir, and J. D. McAuliffe (2017). "Variational Inference: A Review for Statisticians". en. In: *Journal of the American Statistical Association.* (Visited on 2019-02-20).

Bohn, J. and M. Feischl (2021). "Recurrent neural networks as optimal mesh refinement strategies". In: *Computers & Mathematics with Applications* 97, pp. 61–76.

Bommasani, R. (2021). "On the opportunities and risks of foundation models". In: *arXiv preprint arXiv:2108.07258.*

Borouchaki, H., F. Hecht, and P. J. Frey (1998). "Mesh gradation control". In: *International Journal for Numerical Methods in Engineering* 43.6, pp. 1143–1165.

Boyd, S. P. and L. Vandenberghe (2004). *Convex optimization.* Cambridge university press.

Brandstetter, J., D. E. Worrall, and M. Welling (2022). "Message Passing Neural PDE Solvers". In: *International Conference on Learning Representations (ICLR).*

Brenner, S. C. and L. R. Scott (2008). *The mathematical theory of finite element methods*. Vol. 3. Springer.

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba (2016). "Openai gym". In: *arXiv preprint arXiv:1606.01540*.

Bronstein, M. M., J. Bruna, T. Cohen, and P. Velickovic (2021). "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: *CoRR* abs/2104.13478. arXiv: 2104.13478.

Brooks, R. A. (1991). "Intelligence without representation". In: *Artificial intelligence* 47.1-3, pp. 139–159.

Brown, D., W. Goo, P. Nagarajan, and S. Niekum (2019). "Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations". In: *International Conference on Machine Learning (ICML)*. PMLR, pp. 783–792.

Brown, D. S., W. Goo, and S. Niekum (2020). "Better-than-demonstrator imitation learning via automatically-ranked demonstrations". In: *Conference on Robot Learning (CoRL)*. PMLR, pp. 330–359.

Bryan, G. L., M. L. Norman, B. W. O'Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman, et al. (2014). "Enzo: An adaptive mesh refinement code for astrophysics". In: *The Astrophysical Journal Supplement Series* 211.2, p. 19.

Carstensen, C. (2004). "An adaptive mesh-refining algorithm allowing for an H^1 stable L^2 projection onto Courant finite element spaces". In: *Constructive Approximation* 20, pp. 549–564.

Celik, O., A. Taranovic, and G. Neumann (2024). "Acquiring Diverse Skills using Curriculum Reinforcement Learning with Mixture of Experts". In: *Forty-first International Conference on Machine Learning*.

Cerveny, J., V. Dobrev, and T. Kolev (2019). "Nonconforming mesh refinement for high-order finite elements". In: *SIAM Journal on Scientific Computing* 41.4, pp. C367–C392.

Chi, C., Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song (2025). "Diffusion policy: Visuomotor policy learning via action diffusion". In: *The International Journal of Robotics Research* 44.10-11, pp. 1684–1704.

Christiano, P. F., J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei (2017). "Deep reinforcement learning from human preferences". In: *Advances in neural information processing systems* 30.

Chu, X., H. Huang, X. Zhang, F. Wei, and Y. Wang (2025). "Gpg: A simple and strong reinforcement learning baseline for model reasoning". In: *arXiv preprint arXiv:2504.02546*.

Clavier, P., N. Grinsztajn, R. Avalos, Y. Flet-Berliac, I. Ergun, O. D. Domingues, E. Tarassov, O. Pietquin, P. H. Richemond, F. Strub, et al. (2025). "ShiQ: Bringing back Bellman to LLMs". In: *arXiv preprint arXiv:2505.11081*.

Cobbe, K., V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. (2021a). "Training verifiers to solve math word problems". In: *arXiv preprint arXiv:2110.14168*.

Cobbe, K., V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman (2021b). *Training Verifiers to Solve Math Word Problems*. arXiv: 2110.14168 [cs.LG]. URL: https://arxiv.org/abs/2110.14168.

Cohen, A., E. Teng, V.-P. Berges, R.-P. Dong, H. Henry, M. Mattar, A. Zook, and S. Ganguly (2021). "On the use and misuse of absorbing states in multi-agent reinforcement learning". In: *arXiv preprint arXiv:2111.05992.*

Cohere, T., A. Ahmadian, M. Ahmed, J. Alammar, M. Alizadeh, Y. Alnumay, S. Althammer, A. Arkhangorodsky, V. Aryabumi, D. Aumiller, et al. (2025). "Command a: An enterprise-ready large language model". In: *arXiv preprint arXiv:2504.00698.*

Connor, J. J. and C. A. Brebbia (2013). *Finite element techniques for fluid flow.* Newnes.

Cui, G., L. Yuan, Z. Wang, H. Wang, W. Li, B. He, Y. Fan, T. Yu, Q. Xu, W. Chen, J. Yuan, H. Chen, K. Zhang, et al. (2025a). *Process Reinforcement through Implicit Rewards.* arXiv: `2502.01456 [cs.LG]`. URL: `https://arxiv.org/abs/2502.01456`.

Cui, G., Y. Zhang, J. Chen, L. Yuan, Z. Wang, Y. Zuo, H. Li, Y. Fan, H. Chen, W. Chen, et al. (2025b). "The entropy mechanism of reinforcement learning for reasoning language models". In: *arXiv preprint arXiv:2505.22617.*

Cunningham, A. J., A. Frank, P. Varnière, S. Mitran, and T. W. Jones (2009). "Simulating magnetohydrodynamical flow with constrained transport and adaptive mesh refinement: algorithms and tests of the AstroBEAR code". In: *The Astrophysical Journal Supplement Series* 182.2, p. 519.

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4, pp. 303–314.

D'Azevedo, E. F. and R. B. Simpson (1991). "On optimal triangular meshes for minimizing the gradient error". In: *Numerische Mathematik* 59.1, pp. 321–348.

Dahlinger, P., T. Hoang, D. Blessing, N. Freymuth, and G. Neumann (2025). "MaNGO-Adaptable Graph Network Simulators via Meta-Learning". In: *Advances in Neural Information Processing Systems.* Vol. 38.

Daniel, C., G. Neumann, O. Kroemer, J. Peters, et al. (2016). "Hierarchical relative entropy policy search". In: *Journal of Machine Learning Research* 17, pp. 1–50.

Davison, A. C. and D. V. Hinkley (1997). *Bootstrap methods and their application.* Cambridge university press.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1, pp. 1–22.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition.* Ieee, pp. 248–255.

Dontchev, A. L. and R. T. Rockafellar (2009). *Implicit functions and solution mappings.* Vol. 543. Springer.

Dörfler, W. (1996). "A convergent adaptive algorithm for Poisson's equation". In: *SIAM Journal on Numerical Analysis* 33.3, pp. 1106–1124.

Duan, J., H. Cheng, S. Wang, A. Zavalny, C. Wang, R. Xu, B. Kailkhura, and K. Xu (2024). "Shifting Attention to Relevance: Towards the Predictive Uncertainty Quantification of Free-Form Large Language Models". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5050–5063.

Dzanic, T., K. Mittal, D. Kim, J. Yang, S. Petrides, B. Keith, and R. Anderson (2024). "DynAMO: Multi-agent reinforcement learning for dynamic anticipatory mesh optimization with

applications to hyperbolic conservation laws". In: *Journal of Computational Physics*, p. 112924.
Englert, P. and M. Toussaint (2018). "Learning manipulation skills from a single demonstration". In: *The International Journal of Robotics Research* 37.1, pp. 137–154.
Englert, P., N. A. Vien, and M. Toussaint (2017). "Inverse KKT: Learning cost functions of manipulation tasks from demonstrations". In: *The International Journal of Robotics Research* 36.13-14, pp. 1474–1488.
Engstrom, L., A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry (2020). "Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO". In: *International Conference on Learning Representations.* arXiv: 2005.12729. URL: http://arxiv.org/abs/2005.12729.
Ewerton, M., G. Neumann, R. Lioutikov, H. B. Amor, J. Peters, and G. Maeda (2015). "Learning multiple collaborative tasks with a mixture of interaction primitives". In: *2015 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, pp. 1535–1542.
Fedus, W., P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney (2020). "Revisiting fundamentals of experience replay". In: *International Conference on Machine Learning (ICML)*, pp. 3061–3071.
Fidkowski, K. J. and G. Chen (2021). "Metric-based, goal-oriented mesh adaptation using machine learning". In: *Journal of Computational Physics* 426, p. 109957.
Fidkowski, K. J. and D. L. Darmofal (2011). "Review of output-based error estimation and mesh adaptation in computational fluid dynamics". In: *AIAA journal* 49.4, pp. 673–694.
Firoozi, R., J. Tucker, S. Tian, A. Majumdar, J. Sun, W. Liu, Y. Zhu, S. Song, A. Kapoor, K. Hausman, et al. (2025). "Foundation models in robotics: Applications, challenges, and the future". In: *The International Journal of Robotics Research* 44.5, pp. 701–739.
Fish, J. and T. Belytschko (2007). *A first course in finite elements.* Vol. 1. Wiley New York.
Fisher, R. A. (1922). "On the mathematical foundations of theoretical statistics". In: *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character* 222.594-604, pp. 309–368.
Flet-Berliac, Y., N. Grinsztajn, F. Strub, E. Choi, B. Wu, C. Cremer, A. Ahmadian, Y. Chandak, M. G. Azar, O. Pietquin, et al. (2024). "Contrastive policy gradient: Aligning LLMs on sequence-level scores in a supervised-friendly fashion". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 21353–21370.
Floater, M. S. (2003). "Mean value coordinates". In: *Computer Aided Geometric Design* 20.1, pp. 19–27.
Fortunato, M., T. Pfaff, P. Wirnsberger, A. Pritzel, and P. Battaglia (2022). "MultiScale MeshGraphNets". In: *ICML 2022 2nd AI for Science Workshop.* URL: https://openreview.net/forum?id=G3TRIsmMhhf.
Foucart, C., A. Charous, and P. F. Lermusiaux (2023). "Deep reinforcement learning for adaptive mesh refinement". In: *Journal of Computational Physics* 491, p. 112381.
Frey, P. J. and P.-L. George (2007). *Mesh generation: application to finite elements.* Iste. DOI: 10.1002/9780470611166.
Freymuth, N., P. Becker, and G. Neumann (2021). "Versatile Inverse Reinforcement Learning via Cumulative Rewards". In: *Neurips Workshop on Robot Learning: Self-Supervised and Lifelong Learning, Virtual.*

Freymuth, N., P. Dahlinger, T. Würth, S. Reisch, L. Kärger, and G. Neumann (2023). "Swarm reinforcement learning for adaptive mesh refinement". In: *Advances in Neural Information Processing Systems (NeurIPS)* 36.
– (2024). "Adaptive Swarm Mesh Refinement using Deep Reinforcement Learning with Local Rewards". In: *arXiv preprint arXiv:2406.08440*.
Freymuth, N., N. Schreiber, A. Taranovic, P. Becker, and G. Neumann (2022). "Inferring Versatile Behavior from Demonstrations by Matching Geometric Descriptors". In: *6th Annual Conference on Robot Learning*.
Freymuth, N., T. Würth, N. Schreiber, B. Gyenes, A. Boltres, J. Mitsch, A. Taranovic, T. Hoang, P. Dahlinger, P. Becker, L. Kärger, and G. Neumann (2025). "AMBER: Adaptive Mesh Generation by Iterative Mesh Resolution Prediction". In: *Advances in Neural Information Processing Systems*. Vol. 38.
Fu, J., K. Luo, and S. Levine (2018). "Learning Robust Rewards with Adverserial Inverse Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)*.
Gao, B., F. Song, Z. Yang, Z. Cai, Y. Miao, Q. Dong, L. Li, C. Ma, L. Chen, R. Xu, et al. (2025). "Omni-MATH: A Universal Olympiad Level Mathematic Benchmark for Large Language Models". In: *The Thirteenth International Conference on Learning Representations*.
Geuzaine, C. and J.-F. Remacle (2009). "Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities". In: *International Journal for Numerical Methods in Engineering* 79.11, pp. 1309–1331.
Ghasemipour, S. K. S., R. Zemel, and S. Gu (2020). "A divergence minimization perspective on imitation learning methods". In: *Conference on Robot Learning (CoRL)*. PMLR, pp. 1259–1277.
Gibert, G., B. Prabel, A. Gravouil, and C. Jacquemoud (2019). "A 3D automatic mesh refinement X-FEM approach for fatigue crack propagation". In: *Finite Elements in Analysis and Design* 157, pp. 21–37.
Gillette, A., B. Keith, and S. Petrides (2024). "Learning robust marking policies for adaptive mesh refinement". In: *SIAM Journal on Scientific Computing* 46.1, A264–A289.
Gilmer, J., S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl (June 2017). "Neural Message Passing for Quantum Chemistry". In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Proceedings of Machine Learning Research 70. Ed. by D. Precup and Y. W. Teh, pp. 1263–1272. URL: `https://proceedings.mlr.press/v70/gilmer17a.html`.
Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. Vol. 1. `http://www.deeplearningbook.org`. MIT Press.
Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). "Generative adversarial nets". In: *Advances in Neural Information Processing Systems (NIPS)* 27.
Grattafiori, A., A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, A. Fan, et al. (2024). *The Llama 3 Herd of Models*. arXiv: `2407.21783 [cs.AI]`. URL: `https://arxiv.org/abs/2407.21783`.
Guillet, T., R. Pakmor, V. Springel, P. Chandrashekar, and C. Klingenberg (2019). "High-order magnetohydrodynamics for astrophysics with an adaptive mesh refinement discontinuous Galerkin scheme". In: *Monthly Notices of the Royal Astronomical Society* 485.3, pp. 4209–4246.

Guo, X., W. Li, and F. Iorio (2016). "Convolutional Neural Networks for Steady Flow Approximation". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16, pp. 481–490. DOI: 10.1145/2939672.2939738. URL: https://doi.org/10.1145/2939672.2939738.

Gustafsson, T. and G. D. Mcbain (2020). "scikit-fem: A Python package for finite element assembly". In: *Journal of Open Source Software* 5.52, p. 2369.

Haarnoja, T., A. Zhou, P. Abbeel, and S. Levine (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning (ICML)*, pp. 1861–1870.

Hayes, C. F., R. Rădulescu, E. Bargiacchi, J. Källström, M. Macfarlane, M. Reymond, T. Verstraeten, L. M. Zintgraf, R. Dazeley, F. Heintz, et al. (2022). "A practical guide to multi-objective reinforcement learning and planning". In: *Autonomous Agents and Multi-Agent Systems* 36, p. 26.

He, C., R. Luo, Y. Bai, S. Hu, Z. Thai, J. Shen, J. Hu, X. Han, Y. Huang, Y. Zhang, et al. (2024). "OlympiadBench: A Challenging Benchmark for Promoting AGI with Olympiad-Level Bilingual Multimodal Scientific Problems". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850.

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Heaney, D. F. (2018). *Handbook of metal injection molding*. Woodhead Publishing.

Hendrycks, D., C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt (2021). "Measuring Mathematical Problem Solving With the MATH Dataset". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Hernández-Cano, A., A. Hägele, A. H. Huang, A. Romanou, A.-J. Solergibert, B. Pasztor, B. Messmer, D. Garbaya, E. F. Ďurech, I. Hakimi, J. G. Giraldo, M. Ismayilzada, et al. (2025). *Apertus: Democratizing Open and Compliant LLMs for Global Language Environments*. arXiv: 2509.14233 [cs.CL]. URL: https://arxiv.org/abs/2509.14233.

Hessel, M., J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver (2018). "Rainbow: Combining improvements in deep reinforcement learning". In: *Proceedings of the AAAI conference on artificial intelligence* 32.1.

Hinton, G., L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. (2012). "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". In: *IEEE Signal processing magazine* 29.6, pp. 82–97.

Ho, J. and S. Ermon (2016). "Generative Adversarial Imitation Learning". In: *Advances in Neural Information Processing Systems (NIPS)* 29. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf.

Hoang, T., H. Le, P. Becker, V. A. Ngo, and G. Neumann (2025). "Geometry-aware RL for Manipulation of Varying Shapes and Deformable Objects". In: *International Conference on Learning Representations (ICLR)*.

Hochreiter, S. and J. Schmidhuber (Dec. 1997). “Long Short-term Memory”. In: *Neural computation* 9, pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

Hoffmann, J., S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. (2022). “Training compute-optimal large language models”. In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pp. 30016–30030.

Hornik, K. (1991). “Approximation capabilities of multilayer feedforward networks”. In: *Neural networks* 4.2, pp. 251–257.

Hu, J., J. K. Liu, H. Xu, and W. Shen (2025). “REINFORCE++: Stabilizing Critic-Free Policy Optimization with Global Normalization”. In: *arXiv preprint arXiv:2501.03262*.

Hu, P., Y. Wang, and Z.-M. Ma (2024). “Better Neural PDE Solvers Through Data-Free Mesh Movers”. In: *The Twelfth International Conference on Learning Representations (ICLR)*.

Huang, K., M. Krügener, A. Brown, F. Menhorn, H.-J. Bungartz, and D. Hartmann (2021). “Machine learning-based optimal mesh generation in computational fluid dynamics”. In: *arXiv preprint arXiv:2102.12923*.

Huang, S., R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang (2022). “The 37 implementation details of proximal policy optimization”. In: *The ICLR Blog Track 2023*.

Huang, W. and R. D. Russell (2010). *Adaptive moving mesh methods*. Vol. 174. Springer Science & Business Media.

Huergo, D., G. Rubio, and E. Ferrer (2024). “A reinforcement learning strategy for p-adaptation in high order solvers”. In: *Results in Engineering* 21, p. 101693.

HuggingFaceTB (2025). *FineMath-Llama-3B*. https://huggingface.co/HuggingFaceTB/FineMath-Llama-3B. Hugging Face model card; licensed under Apache-2.0.

Hughes, T. J. (2003). *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation.

Hussein, A., M. M. Gaber, E. Elyan, and C. Jayne (2017). “Imitation learning: A survey of learning methods”. In: *ACM Computing Surveys (CSUR)* 50.2, pp. 1–35.

Hüttenrauch, M., Š. Adrian, and G. Neumann (2019). “Deep reinforcement learning for swarm systems”. In: *Journal of Machine Learning Research* 20.54, pp. 1–31.

Intelligence, P., A. Amin, R. Aniceto, A. Balakrishna, K. Black, K. Conley, G. Connors, J. Darpinian, K. Dhabalia, J. DiCarlo, et al. (2025). “$\pi^*_{0.6}$: A VLA That Learns From Experience”. In: *arXiv preprint arXiv:2511.14759*.

Ioffe, S. and C. Szegedy (2015). “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning (ICML)*, pp. 448–456.

Jian, Y., L. Hongqiang, X. Ran, L. Xuejun, et al. (2025). “Para2Mesh: A dual diffusion framework for moving mesh adaptation”. In: *Chinese Journal of Aeronautics*, p. 103441.

Jin, J.-M. (2015). *The finite element method in electromagnetics*. John Wiley & Sons.

John, V. et al. (2016). *Finite element methods for incompressible flow problems*. Vol. 51. Springer.

Jones, M. T. and P. E. Plassmann (1997). “Adaptive refinement of unstructured finite-element meshes”. In: *Finite Elements in Analysis and Design* 25.1-2, pp. 41–60.

Kakade, S. (2001). “A Natural Policy Gradient”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 14.

Kakade, S. and J. Langford (2002a). “Approximately Optimal Approximate Reinforcement Learning”. In: *International Conference on Machine Learning*.

Kakade, S. M. and J. C. Langford (2002b). “Approximately Optimal Approximate Reinforcement Learning”. In: *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 267–274. URL: https://dl.acm.org/doi/10.5555/645531.656005.

Kaplan, J., S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei (2020). “Scaling laws for neural language models”. In: *arXiv preprint arXiv:2001.08361*.

Karush, W. (1939). “Minima of functions of several variables with inequalities as side constraints”. In: *M. Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago*.

Khan, A., M. Yamada, A. Chikane, and M. Kaul (2024). “GraphMesh: Geometrically Generalized Mesh Refinement Using GNNs”. In: *International Conference on Computational Science*, pp. 120–134.

Kim, M., J. Lee, and J. Kim (2023). “GMR-Net: GCN-based mesh refinement framework for elliptic PDE problems”. In: *Engineering with Computers* 39.5, pp. 3721–3737.

Kim, M. J., K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. P. Foster, P. R. Sanketi, Q. Vuong, et al. (2024). “OpenVLA: An Open-Source Vision-Language-Action Model”. In: *8th Annual Conference on Robot Learning*.

Kingma, D. P. and J. Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations (ICLR)*. Ed. by Y. Bengio and Y. LeCun. URL: http://arxiv.org/abs/1412.6980.

Kipf, T. N. and M. Welling (2017). “Semi-Supervised Classification with Graph Convolutional Networks”. In: *International Conference on Learning Representations (ICLR)*. URL: https://openreview.net/forum?id=SJU4ayYgl.

Kita, E. and N. Kamiya (2001). “Error estimation and adaptive mesh refinement in boundary element method, an overview”. In: *Engineering Analysis with Boundary Elements* 25.7, pp. 479–495.

Koch, S., A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, and D. Panozzo (2019). “ABC: A big CAD model dataset for geometric deep learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9601–9611.

Kostrikov, I., K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson (2018). “Discriminator-Actor-Critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning”. In: *International Conference on Learning Representations (ICLR)*.

Kostrikov, I., O. Nachum, and J. Tompson (2019). “Imitation learning via off-policy distribution matching”. In: *arXiv preprint arXiv:1912.05032*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25.

Kullback, S. and R. A. Leibler (1951). “On Information and Sufficiency”. In: *The Annals of Mathematical Statistics* 22.1, pp. 79–86.

Kunstner, F., A. Milligan, R. Yadav, M. Schmidt, and A. Bietti (2024). “Heavy-tailed class imbalance and why adam outperforms gradient descent on language models”. In: *Advances in Neural Information Processing Systems* 37, pp. 30106–30148.

Lam, R., A. Sanchez-Gonzalez, M. Willson, P. Wirnsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu, et al. (2023). "Learning skillful medium-range global weather forecasting". In: *Science* 382.6677, pp. 1416–1421.

Lambert, N., J. Morrison, V. Pyatkin, S. Huang, H. Ivison, F. Brahman, L. J. V. Miranda, A. Liu, N. Dziri, S. Lyu, Y. Gu, S. Malik, et al. (2024). "Tulu 3: Pushing frontiers in open language model post-training". In: *arXiv preprint arXiv:2411.15124*.

Larson, M. G. and F. Bengzon (2013). *The Finite Element Method: Theory, Implementation, and Applications*. Vol. 10. Texts in Computational Science and Engineering. Berlin, Heidelberg: Springer. ISBN: 978-3-642-33286-9 978-3-642-33287-6. DOI: 10.1007/978-3-642-33287-6. (Visited on 2023-10-25).

Ho-Le, K. (1988). "Finite element mesh generation methods: a review and classification". In: *Computer-aided design* 20.1, pp. 27–38.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (2002). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lee, D.-H. et al. (2013). "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks". In: *Workshop on challenges in representation learning, ICML*. Vol. 3. Atlanta, p. 896.

Lee, H., D. Hwang, D. Kim, H. Kim, J. J. Tai, K. Subramanian, P. R. Wurman, J. Choo, P. Stone, and T. Seno (2024). "SimBa: Simplicity Bias for Scaling Up Parameters in Deep Reinforcement Learning". In: *The Thirteenth International Conference on Learning Representations*.

Lestringant, C., B. Audoly, and D. M. Kochmann (2020). "A discrete, geometrically exact method for simulating nonlinear, elastic and inelastic beams". In: *Computer Methods in Applied Mechanics and Engineering* 361, p. 112741.

Lewkowycz, A., A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, et al. (2022). "Solving quantitative reasoning problems with language models". In: *Advances in neural information processing systems* 35, pp. 3843–3857.

Li, G., H. Zhou, D. Roth, S. Thilges, F. Otto, R. Lioutikov, and G. Neumann (2024a). *Open the Black Box: Step-based Policy Upyears for Temporally-Correlated Episodic Reinforcement Learning*. arXiv: 2401.11437 [cs.LG].

Li, J., E. Beeching, L. Tunstall, B. Lipkin, R. Soletskyi, S. Huang, K. Rasul, L. Yu, A. Q. Jiang, Z. Shen, et al. (2024b). "Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions". In: *Hugging Face repository* 13.9, p. 9.

Li, R., J. Fu, B.-W. Zhang, T. Huang, Z. Sun, C. Lyu, G. Liu, Z. Jin, and G. Li (2023). "Taco: Topics in algorithmic code generation dataset". In: *arXiv preprint arXiv:2312.14852*.

Li, Y., D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. de Masson d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals (2022). "Competition-level code generation with AlphaCode". In: *Science* 378.6624, pp. 1092–1097. DOI: 10.1126/science.abq1158. eprint: https://www.science.org/doi/pdf/10.1126/science.abq1158. URL: https://www.science.org/doi/abs/10.1126/science.abq1158.

Lin, L.-J. (1992). "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine Learning* 8, pp. 293–321.
Linkerhägner, J., N. Freymuth, P. M. Scheikl, F. Mathis-Ullrich, and G. Neumann (2023). "Grounding Graph Network Simulators using Physical Sensor Observations". In: *The Eleventh International Conference on Learning Representations (ICLR)*. URL: https://openreview.net/forum?id=jsZsEd8VEY.
Lipman, Y., R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le (2023). "Flow Matching for Generative Modeling". In: *The Eleventh International Conference on Learning Representations*.
Liu, W. K., S. Li, and H. S. Park (2022). "Eighty years of the finite element method: Birth, evolution, and future". In: *Archives of Computational Methods in Engineering*, pp. 1–23.
Liu, Z., C. Chen, W. Li, P. Qi, T. Pang, C. Du, W. S. Lee, and M. Lin (2025). "Understanding r1-zero-like training: A critical perspective". In: *arXiv preprint arXiv:2503.20783*.
Lo, D. S. (2014). *Finite element mesh generation*. CRC press.
Lock, C., O. Hassan, R. Sevilla, and J. Jones (2023a). "Meshing using neural networks for improving the efficiency of computer modelling". In: *Engineering with Computers* 39.6, pp. 3791–3820.
– (2023b). "Predicting the Near-Optimal Mesh Spacing for a Simulation Using Machine Learning". In: *International Meshing Roundtable*, pp. 115–136.
Lopez-Guevara, T., Y. Rubanova, W. F. Whitney, T. Pfaff, K. Stachenfeld, and K. R. Allen (2024). "Scaling Face Interaction Graph Networks to Real World Scenes". In: *arXiv preprint arXiv:2401.11985*.
Loseille, A. and F. Alauzet (2011). "Continuous mesh framework part I: well-posed continuous interpolation error". In: *SIAM Journal on Numerical Analysis* 49.1, pp. 38–60.
Magnus, J. R. and H. Neudecker (1989). "Matrix differential calculus". In: *Econom. Theor* 5, pp. 161–165.
Mandlekar, A., D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín (2021). "What Matters in Learning from Offline Human Demonstrations for Robot Manipulation". In: *5th Annual Conference on Robot Learning*.
Manschitz, S., J. Kober, M. Gienger, and J. Peters (2014). "Learning to sequence movement primitives from demonstrations". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 4414–4421.
Marcum, D. and F. Alauzet (2014). "Aligned metric-based anisotropic solution adaptive mesh generation". In: *Procedia Engineering* 82, pp. 428–444.
Mildenhall, B., P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng (2020). "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *European Conference on Computer Vision (ECCV)*, pp. 405–421.
Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning (ICML)*, pp. 1928–1937.
Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller (2013). "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602*.
Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou,

H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (2015). "Human-level control through deep reinforcement learning". en. In: *Nature* 518.7540, pp. 529–533. (Visited on 2018-07-16).

Mukherjee, A. (1996). *An adaptive finite element code for elliptic boundary value problems in three dimensions with applications in numerical relativity.* The Pennsylvania State University.

Mülling, K., J. Kober, O. Kroemer, and J. Peters (2013). "Learning to select and generalize striking movements in robot table tennis". In: *The International Journal of Robotics Research* 32.3, pp. 263–279.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* MIT press.

Nagarajan, A. and S. Soghrati (2018). "Conforming to interface structured adaptive mesh refinement: 3D algorithm and implementation". In: *Computational Mechanics* 62, pp. 1213–1238.

Nauman, M. and M. Cygan (2024a). "On the Theory of Risk-Aware Agents: Bridging Actor-Critic and Economics". In: *ICML 2024 Workshop: Aligning Reinforcement Learning Experimentalists and Theorists.*

Nauman, M., M. Ostaszewski, K. Jankowski, P. Miłoś, and M. Cygan (2024b). "Bigger, regularized, optimistic: scaling for compute and sample efficient continuous control". In: *Advances in neural information processing systems* 37, pp. 113038–113071.

Neal, R. M. and G. E. Hinton (1998). "A view of the EM algorithm that justifies incremental, sparse, and other variants". In: *Learning in graphical models.* Springer, pp. 355–368.

Nemec, M., M. Aftosmis, and M. Wintzer (2008). "Adjoint-based adaptive mesh refinement for complex geometries". In: *46th AIAA Aerospace Sciences Meeting and Exhibit*, p. 725.

Neumann, G., W. Maass, and J. Peters (2009). "Learning complex motions by sequencing simpler motion templates". In: *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 753–760.

Nguyen, T., A. Koneru, S. Li, and A. Grover (2025). "Physix: A foundation model for physics simulations". In: *arXiv preprint arXiv:2506.17774.*

Orsini, M., A. Raichuk, L. Hussenot, D. Vincent, R. Dadashi, S. Girgin, M. Geist, O. Bachem, O. Pietquin, and M. Andrychowicz (2021). "What Matters for Adversarial Imitation Learning?" In: *Advances in Neural Information Processing Systems (NeurIPS)* 34. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, pp. 14656–14668. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/7b647a7d88f4d6319bf0d600d168dbeb-Paper.pdf.

Ortiz, M. and J. Quigley Iv (1991). "Adaptive mesh refinement in strain localization problems". In: *Computer Methods in Applied Mechanics and Engineering* 90.1-3, pp. 781–804.

Osa, T., A. M. G. Esfahani, R. Stolkin, R. Lioutikov, J. Peters, and G. Neumann (2017). "Guiding trajectory optimization by demonstrated distributions". In: *IEEE Robotics and Automation Letters* 2.2, pp. 819–826.

Osa, T., J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, et al. (2018). "An algorithmic perspective on imitation learning". In: *Foundations and Trends® in Robotics* 7.1-2, pp. 1–179.

Otto, F., P. Becker, V. A. Ngo, H. C. M. Ziesche, and G. Neumann (2021). *Differentiable Trust Region Layers for Deep Reinforcement Learning*. URL: https://openreview.net/forum?id=qYZD-AO1Vn.

Otto, F., P. Becker, N. A. Vien, and G. Neumann (2025). "Efficient Off-Policy Learning for High-Dimensional Action Spaces". In: *International Conference on Learning Representations*.

Ouyang, L., J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, et al. (2022). "Training language models to follow instructions with human feedback". In: *Advances in neural information processing systems* 35, pp. 27730–27744.

Pan, J., J. Huang, G. Cheng, and Y. Zeng (2023). "Reinforcement learning for automatic quadrilateral mesh generation: A soft actor–critic approach". In: *Neural Networks* 157, pp. 288–304.

Paraschos, A., C. Daniel, J. R. Peters, and G. Neumann (2013). "Probabilistic movement primitives". In: *Advances in neural information processing systems (NIPS)* 26.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems (NeurIPS)* 32.

Pervez, A. and D. Lee (2018). "Learning task-parameterized dynamic movement primitives using mixture of GMMs". In: *Intelligent Service Robotics* 11.1, pp. 61–78.

Peters, J., K. Mulling, and Y. Altun (2010). "Relative entropy policy search". In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Pfaff, T., M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia (2021). "Learning Mesh-Based Simulation with Graph Networks". In: *International Conference on Learning Representations (ICLR)*.

Piantadosi, S. T. (2014). "Zipf's word frequency law in natural language: A critical review and future directions". In: *Psychonomic bulletin & review* 21.5, pp. 1112–1130.

Plewa, T., T. Linde, V. G. Weirs, et al. (2005). *Adaptive mesh refinement-theory and applications*. Springer.

Price, I., A. Sanchez-Gonzalez, F. Alet, T. R. Andersson, A. El-Kadi, D. Masters, T. Ewalds, J. Stott, S. Mohamed, P. Battaglia, et al. (2025). "Probabilistic weather forecasting with machine learning". In: *Nature* 637.8044, pp. 84–90.

Qi, C. R., H. Su, K. Mo, and L. J. Guibas (2017a). "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 652–660.

Qi, C. R., L. Yi, H. Su, and L. J. Guibas (2017b). "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Advances in Neural Information Processing Systems (NIPS)* 30. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. URL: https://proceedings.neurips.cc/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf.

Quarteroni, A. and S. Quarteroni (2009). *Numerical models for differential problems*. Vol. 2. Springer.

Radford, A., J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. (2021). "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PmLR, pp. 8748–8763.

Radford, A., J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever (2019). "Language Models are Unsupervised Multitask Learners". In: *arXiv*.

Rafailov, R., A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn (2023). "Direct preference optimization: Your language model is secretly a reward model". In: *Advances in neural information processing systems* 36, pp. 53728–53741.

Raffin, A., A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann (2021). "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research (JMLR)* 22.268, pp. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html`.

Raissi, M., P. Perdikaris, and G. E. Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational physics* 378, pp. 686–707.

Reddy, J. N. (2019). *Introduction to the finite element method*. McGraw-Hill Education.

Remacle, J.-F., F. Henrotte, T. Carrier-Baudouin, E. Béchet, E. Marchandise, C. Geuzaine, and T. Mouton (2013). "A frontal Delaunay quad mesh generator using the L-infinity norm". In: *International Journal for Numerical Methods in Engineering* 94.5, pp. 494–512.

Ren, Y. and D. J. Sutherland (2024). "Learning Dynamics of LLM Finetuning". In: *The Thirteenth International Conference on Learning Representations*.

Reuss, M., H. Zhou, M. Rühle, Ö. E. Yağmurlu, F. Otto, and R. Lioutikov (27–30 Sep 2025). "FLOWER: Democratizing Generalist Robot Policies with Efficient Vision-Language-Flow Models". In: *Proceedings of The 9th Conference on Robot Learning*. Ed. by J. Lim, S. Song, and H.-W. Park. Vol. 305. Proceedings of Machine Learning Research. PMLR, pp. 3736–3761.

Richemond, P. H., Y. Tang, D. Guo, D. Calandriello, M. G. Azar, R. Rafailov, B. A. Pires, E. Tarassov, L. Spangher, W. Ellsworth, et al. (2024). "Offline regularised reinforcement learning for large language models alignment". In: *arXiv preprint arXiv:2405.19107*.

Rivara, M. C. (1984). "Algorithms for refining triangular grids suitable for adaptive and multigrid techniques". In: *International journal for numerical methods in Engineering* 20.4, pp. 745–756.

Rong, Y., W. Huang, T. Xu, and J. Huang (2019). "DropEdge: Towards Deep Graph Convolutional Networks on Node Classification". In: *International Conference on Learning Representations (ICLR)*.

Ronneberger, O., P. Fischer, and T. Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241.

Rosato, D. V. and M. G. Rosato (2012). *Injection molding handbook*. Springer Science & Business Media.

Ross, S., G. Gordon, and D. Bagnell (Nov. 2011). "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*. Proceedings of

Machine Learning Research 15. Ed. by G. Gordon, D. Dunson, and M. Dudík, pp. 627–635. URL: https://proceedings.mlr.press/v15/ross11a.html.
Roth, J., M. Schröder, and T. Wick (2022). "Neural network guided adjoint computations in dual weighted residual error estimation". In: *SN Applied Sciences* 4.2, p. 62.
Ruan, Y., C. J. Maddison, and T. B. Hashimoto (2024). "Observational scaling laws and the predictability of langauge model performance". In: *Advances in Neural Information Processing Systems* 37, pp. 15841–15892.
Sanchez-Gamero, S., O. Hassan, and R. Sevilla (2024). "A machine learning approach to predict near-optimal meshes for turbulent compressible flow simulations". In: *International Journal of Computational Fluid Dynamics* 38.2-3, pp. 221–245.
Sanchez-Gonzalez, A., J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia (2020). "Learning to Simulate Complex Physics with Graph Networks". In: *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pp. 8459–8468.
Scarselli, F., M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini (2009). "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1, pp. 61–80. DOI: 10.1109/TNN.2008.2005605.
Schaal, S. (1996). "Learning from demonstration". In: *Advances in Neural Information Processing Systems (NeurIPS)* 9.
– (2006). "Dynamic movement primitives - a framework for motor control in humans and humanoid robotics". In: *Adaptive motion of animals and machines*. Springer, pp. 261–280.
Schaul, T., J. Quan, I. Antonoglou, and D. Silver (2016). "Prioritized Experience Replay". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun.
Scheikl, P. M., N. Schreiber, C. Haas, N. Freymuth, G. Neumann, R. Lioutikov, and F. Mathis-Ullrich (2024). "Movement primitive diffusion: Learning gentle robotic manipulation of deformable objects". In: *IEEE Robotics and Automation Letters* 9.6, pp. 5338–5345.
Schulman, J., S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz (2015a). "Trust Region Policy Optimization". In: *International Conference on Machine Learning*.
Schulman, J., P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel (2015b). "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *arXiv preprint arXiv:1506.02438*.
Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). "Proximal Policy Optimization Algorithms". In: *arXiv preprint arXiv:1707.06347*.
Shao, Z., P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo (2024). "Deepseekmath: Pushing the limits of mathematical reasoning in open language models". In: *arXiv preprint arXiv:2402.03300*.
Shimada, K. (2006). "Current trends and issues in automatic mesh generation". In: *Computer-Aided Design and Applications* 3.6, pp. 741–750.
Shorten, C. and T. M. Khoshgoftaar (2019). "A survey on image data augmentation for deep learning". In: *Journal of Big Data* 6.1, pp. 1–48.
Shumailov, I., Z. Shumaylov, Y. Zhao, N. Papernot, R. Anderson, and Y. Gal (2024). "AI models collapse when trained on recursively generated data". In: *Nature* 631.8022, pp. 755–759.
Si, H. (2008). "Adaptive tetrahedral mesh generation by constrained Delaunay refinement". In: *International Journal for Numerical Methods in Engineering* 75.7, pp. 856–880.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. (2017). “Mastering the game of go without human knowledge”. In: *nature* 550.7676, pp. 354–359.

Silver, D., S. Singh, D. Precup, and R. S. Sutton (2021). “Reward is enough”. In: *Artificial intelligence* 299, p. 103535.

Służalec, T., R. Grzeszczuk, S. Rojas, W. Dzwinel, and M. Paszyński (2023). “Quasi-optimal hp-finite element refinements towards singularities via deep neural network prediction”. In: *Computers & Mathematics with Applications* 142, pp. 157–174.

Smith, M. (2009). *ABAQUS/Standard User's Manual, Version 6.9*. English. United States: Dassault Systèmes Simulia Corp.

Song, H. F., A. Abdolmaleki, J. T. Springenberg, A. Clark, H. Soyer, J. W. Rae, S. Noury, A. Ahuja, S. Liu, D. Tirumala, N. Heess, D. Belov, M. Riedmiller, and M. M. Botvinick (2020). “V-MPO: On-Policy Maximum a Posteriori Policy Optimization for Discrete and Continuous Control”. In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SylOlp4FvH.

Song, W., M. Zhang, J. G. Wallwork, J. Gao, Z. Tian, F. Sun, M. Piggott, J. Chen, Z. Shi, X. Chen, et al. (2022). “M2N: Mesh movement networks for PDE solvers”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 35, pp. 7199–7210.

Song, Y., J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole (2021). “Score-Based Generative Modeling through Stochastic Differential Equations”. In: *International Conference on Learning Representations*.

Šošić, A., W. R. KhudaBukhsh, A. M. Zoubir, and H. Koeppl (2017). “Inverse Reinforcement Learning in Swarm Systems”. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pp. 1413–1421.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014-01). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research (JMLR)* 15.1, pp. 1929–1958. ISSN: 1532-4435.

Stein, E. (2007). *Adaptive finite elements in linear and nonlinear solid and structural mechanics*. Vol. 416. Springer Science & Business Media.

Stevenson, R. (2008). “The completion of locally refined simplicial partitions created by bisection”. In: *Mathematics of computation* 77.261, pp. 227–241.

Stiennon, N., L. Ouyang, J. Wu, D. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano (2020). “Learning to summarize with human feedback”. In: *Advances in neural information processing systems* 33, pp. 3008–3021.

Strauss, W. A. (2007). *Partial differential equations: An introduction*. John Wiley & Sons.

Suárez, J. P., P. Abad, A. Plaza, and M. A. Padron (2005). “Computational aspects of the refinement of 3D tetrahedral meshes”. In: *Journal of Computational Methods in Sciences and Engineering* 5.4, pp. 215–224.

Sugiyama, M., T. Suzuki, and T. Kanamori (2012). *Density Ratio Estimation in Machine Learning*. Cambridge University Press.

Suk, J., B. Imre, and J. M. Wolterink (2024). “LaB-GATr: geometric algebra transformers for large biomedical surface and volume meshes”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, pp. 185–195.

Sunehag, P., G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, et al. (2017). "Value-decomposition networks for cooperative multi-agent learning". In: *arXiv preprint arXiv:1706.05296*.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems* 27.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., D. McAllester, S. Singh, and Y. Mansour (1999). "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. (Visited on 2018-08-01).

Tobin, J., R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel (2017). "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, pp. 23–30.

Torabi, F., G. Warnell, and P. Stone (2019). "Adversarial imitation learning from state-only demonstrations". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2229–2231.

Trung, L., X. Zhang, Z. Jie, P. Sun, X. Jin, and H. Li (Aug. 2024). "ReFT: Reasoning with Reinforced Fine-Tuning". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by L.-W. Ku, A. Martins, and V. Srikumar. Bangkok, Thailand: Association for Computational Linguistics, pp. 7601–7614. DOI: 10.18653/v1/2024.acl-long.410. URL: https://aclanthology.org/2024.acl-long.410/.

Um, K., X. Hu, and N. Thuerey (2018). "Liquid Splash Modeling with Neural Networks". In: *Computer Graphics Forum* 37.8, pp. 171–182. DOI: https://doi.org/10.1111/cgf.13522. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13522. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13522.

Van Hasselt, H., A. Guez, and D. Silver (2016). "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI conference on artificial intelligence* 30.1.

Van Moffaert, K. and A. Nowé (2014). "Multi-objective reinforcement learning using sets of pareto dominating policies". In: *The Journal of Machine Learning Research* 15.1, pp. 3483–3512.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems (NeurIPS)* 30. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Velickovic, P., G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio (2018). "Graph Attention Networks". In: *stat* 1050, p. 4.

Villalobos, P., A. Ho, J. Sevilla, T. Besiroglu, L. Heim, and M. Hobbhahn (2024). "Position: Will we run out of data? Limits of LLM scaling based on human-generated data". In: *Forty-first International Conference on Machine Learning*.

Vinyals, O., I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *nature* 575.7782, pp. 350–354.

Wallwork, J. G. (2021). "Mesh adaptation and adjoint methods for finite element coastal ocean modelling". PhD thesis. Imperial College London.

Wallwork, J. G., J. Lu, M. Zhang, and M. D. Piggott (2022). "E2N: Error Estimation Networks for Goal-Oriented Mesh Adaptation". In: *arXiv preprint arXiv:2207.11233*.

Wang, S., S. Toyer, A. Gleave, and S. Emmons (2020a). *The `imitation` Library for Imitation Learning and Inverse Reinforcement Learning*. `https://github.com/HumanCompatibleAI/imitation`.

Wang, Y., H. He, and X. Tan (22–25 Jul 2020b). "Truly Proximal Policy Optimization". In: *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*. Ed. by R. P. Adams and V. Gogate. Vol. 115. Proceedings of Machine Learning Research. PMLR, pp. 113–122. URL: `https://proceedings.mlr.press/v115/wang20b.html`.

Wang, Y., H. He, X. Tan, and Y. Gan (2019). "Trust region-guided proximal policy optimization". In: *Advances in Neural Information Processing Systems* 32.

Wang, Z., T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas (2016). "Dueling network architectures for deep reinforcement learning". In: *International conference on machine learning (ICML)*, pp. 1995–2003.

Wanner, G. and E. Hairer (1996). *Solving ordinary differential equations II*. Vol. 375. Springer Berlin Heidelberg New York.

Wiesner, F., M. Wessling, and S. Baek (2025). "Towards a physics foundation model". In: *arXiv preprint arXiv:2509.13805*.

Williams, R. J. (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning*.

Wu, T., T. Maruyama, Q. Zhao, G. Wetzstein, and J. Leskovec (2023). "Learning Controllable Adaptive Simulation for Multi-resolution Physics". In: *The Eleventh International Conference on Learning Representations (ICLR)*. URL: `https://openreview.net/forum?id=PbfgkZ2HdbE`.

Wu, T., L. Pan, J. Zhang, T. Wang, Z. Liu, and D. Lin (2021). "Density-aware chamfer distance as a comprehensive metric for point cloud completion". In: *Proceedings of the 35th International Conference on Neural Information Processing Systems (NeurIPS)*. NIPS '21.

Würth, T., N. Freymuth, G. Neumann, and L. Kärger (2025). "Diffusion-Based Hierarchical Graph Neural Networks for Simulating Nonlinear Solid Mechanics". In: *Advances in Neural Information Processing Systems*. Vol. 38.

Würth, T., N. Freymuth, C. Zimmerling, G. Neumann, and L. Kärger (2024). "Physics-informed MeshGraphNets (PI-MGNs): Neural finite element solvers for non-stationary and nonlinear simulations on arbitrary meshes". In: *Computer Methods in Applied Mechanics and Engineering* 429, p. 117102.

Würth, T., C. Krauß, C. Zimmerling, and L. Kärger (2023). "Physics-informed neural networks for data-free surrogate modelling and engineering optimization–An example from composite manufacturing". In: *Materials & Design* 231, p. 112034.

Xi, Z., X. Guo, Y. Nan, E. Zhou, J. Shen, W. Chen, J. Liu, J. Huang, Z. Zhang, H. Guo, et al. (2025). "BAPO: Stabilizing Off-Policy Reinforcement Learning for LLMs via Balanced Policy Optimization with Adaptive Clipping". In: *arXiv preprint arXiv:2510.18927*.

Xie, Q., M.-T. Luong, E. Hovy, and Q. V. Le (2020). “Self-training with noisy student improves imagenet classification”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10687–10698.

Yang, A., A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, et al. (2025a). *Qwen3 Technical Report.* arXiv: `2505.09388 [cs.CL]`. URL: `https://arxiv.org/abs/2505.09388`.

Yang, A., B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, et al. (2025b). *Qwen2.5 Technical Report.* arXiv: `2412.15115 [cs.CL]`. URL: `https://arxiv.org/abs/2412.15115`.

Yang, J., T. Dzanic, B. K. Petersen, J. Kudo, K. Mittal, V. Tomov, J.-S. Camier, T. Zhao, H. Zha, T. Kolev, R. Anderson, and D. Faissol (2023a). “Reinforcement learning for Adaptive Mesh Refinement”. In: *International Conference on Artificial Intelligence and Statistics (AISTATS).*

Yang, J., K. Mittal, T. Dzanic, S. Petrides, B. Keith, B. Petersen, D. Faissol, and R. Anderson (2023b). “Multi-Agent Reinforcement Learning for Adaptive Mesh Refinement”. In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS).*

Yano, M. and D. L. Darmofal (2012). “An optimization-based framework for anisotropic simplex mesh adaptation”. In: *Journal of Computational Physics* 231.22, pp. 7626–7649.

Yu, J., H. Lyu, R. Xu, W. Ouyang, and X. Liu (2024). “Flow2Mesh: A flow-guided data-driven mesh adaptation framework”. In: *Physics of Fluids* 36.3.

Yu, Q., Z. Zhang, R. Zhu, Y. Yuan, X. Zuo, Y. Yue, W. Dai, T. Fan, G. Liu, L. Liu, X. Liu, H. Lin, et al. (2025). *DAPO: An Open-Source LLM Reinforcement Learning System at Scale.* arXiv: `2503.14476 [cs.LG]`. URL: `https://arxiv.org/abs/2503.14476`.

Zhang, H., M. Cisse, Y. N. Dauphin, and D. Lopez-Paz (2018). “mixup: Beyond Empirical Risk Minimization”. In: *International Conference on Learning Representations.*

Zhang, M., C. Wang, S. Kramer, J. G. Wallwork, S. Li, J. Liu, X. Chen, and M. D. Piggott (2024). “UM2N: Towards Universal Mesh Movement Networks”. In: *arXiv e-prints*, arXiv–2407.

Zhang, X., Y. Li, Z. Zhang, and Z.-L. Zhang (2020a). “f-gail: Learning f-divergence for generative adversarial imitation learning”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 33, pp. 12805–12815.

Zhang, Z., P. K. Jimack, and H. Wang (July 2021). “MeshingNet3D: Efficient Generation of Adapted Tetrahedral Meshes for Computational Mechanics”. In: *Advances in Engineering Software* 157–158, p. 103021. ISSN: 0965-9978. DOI: `10.1016/j.advengsoft.2021.103021`. (Visited on 2024-05-22).

Zhang, Z., Y. Wang, P. K. Jimack, and H. Wang (2020b). “MeshingNet: A new mesh generation method based on deep learning”. In: *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part III 20*, pp. 186–198.

Zheng, C., S. Liu, M. Li, X.-H. Chen, B. Yu, C. Gao, K. Dang, Y. Liu, R. Men, A. Yang, J. Zhou, and J. Lin (2025). “Group sequence policy optimization”. In: *arXiv preprint arXiv:2507.18071.*

Zhou, C., Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He, et al. (2025). “A comprehensive survey on pretrained foundation models: A history from bert to chatgpt”. In: *International Journal of Machine Learning and Cybernetics* 16.12, pp. 9851–9915.

Zhou, Y., J. Gao, and T. Asfour (2020). "Movement primitive learning and generalization: Using mixture density networks". In: *IEEE Robotics & Automation Magazine* 27.2, pp. 22–32.

Ziebart, B. D., A. L. Maas, J. A. Bagnell, and A. K. Dey (2008). "Maximum entropy inverse reinforcement learning." In: *Aaai.* Vol. 8. Chicago, IL, USA, pp. 1433–1438.

Ziegler, D. M., N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving (2019). "Fine-tuning language models from human preferences". In: *arXiv preprint arXiv:1909.08593*.

Zienkiewicz, O. C. and R. L. Taylor (2005). *The finite element method for solid and structural mechanics*. Elsevier.

Zienkiewicz, O. C. and J. Z. Zhu (1992). "The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique". In: *International Journal for Numerical Methods in Engineering* 33.7, pp. 1331–1364.

Zimmerling, C., D. Dörr, F. Henning, and L. Kärger (Sept. 2019a). "A Machine Learning Assisted Approach for Textile Formability Assessment and Design Improvement of Composite Components". In: *Compos Part A* 124, p. 105459. ISSN: 1359-835X. DOI: 10.1016/j.compositesa.2019.05.027. (Visited on 2024-03-05).

Zimmerling, C., C. Poppe, O. Stein, and L. Kärger (Feb. 2022). "Optimisation of Manufacturing Process Parameters for Variable Component Geometries Using Reinforcement Learning". In: *Materials & Design* 214, p. 110423. ISSN: 02641275. DOI: 10.1016/j.matdes.2022.110423. (Visited on 2023-11-22).

Zimmerling, C., D. Trippe, B. Fengler, and L. Kärger (July 2019b). "An approach for rapid prediction of textile draping results for variable composite component geometries using deep neural networks". In: *AIP Conference Proceedings* 2113.1. 020007. ISSN: 0094-243X. DOI: 10.1063/1.5112512. eprint: https://pubs.aip.org/aip/acp/article-pdf/doi/10.1063/1.5112512/13178155/020007_1_online.pdf. URL: https://doi.org/10.1063/1.5112512.

Zipf, G. K. (1949). *Human behavior and the principle of least effort: An introduction to human ecology*. Ravenio books.

A. Appendix for Chapter 3

A.1. Additional Experiments

We perform additional experiments for *VIGOR* and *EM+D-REX* on the `Planar Reacher` and `Panda Reacher` tasks. Overall, we study the choice of geometric descriptors, trying both a concatenation of joint angles and context (*Angles*) as well as distances and velocity and acceleration for each robot joint (*Jointwise*). Additionally, we look at the performance of *VIGOR* when trained directly on human demonstrations that are sup-sampled to T time-steps (*Raw*) without first fitting a ProMP on the said demonstrations. We consider the discriminator's architecture, trying out both a shared MLP per step (*MLP*) with an additional encoding of the time-step, and a Long Short-Term Memory (*LSTM*) (Hochreiter et al., 1997). Both architectures are configured to be of similar size to the $1d$-CNN. We also evaluate the effects of different numbers of mixture components (*#Components*), using an ensemble of discriminators (*#Discriminators*), the stepwise BCE loss (*Discriminator Loss* of Equation 3.3, and the amount of training contexts (*#Contexts*). Finally, we inspect the number of used demonstrations per context (*#Demonstrations)*, and the number of $1d$-CNN layers (*#Layers*) as well as the number of convolutional channels per layer (*#Channels*). For *EM+D-REX*, we look at how the number of used EM components influences the performance, and how the algorithm-specific hyperparameters (see Section 3.4.1) need to be tuned for good performance.

Table 3.1 and Table A.1 show `Planar Reacher` results for *VIGOR* and *EM+D-REX*, respectively. `Panda Reacher` results are provided in Table A.2 and Table A.3. All figures sort components by their performance, i.e., the first component c_0 is the best-performing one. In the main experiments, we evaluate the performance of the best component, i.e., c_0.

These experiments show that *EM+D-REX* performs well if tuned right, but that it is very sensitive to the choice of hyperparameters. The most important choice of hyperparameter seems to be the number of EM components compared to the number of demonstrations per training context. Depending on the task and the remaining parameters, *EM+D-REX* either prefers a number of components similar to the number of demonstrations, or only a single component. We assume that this sensitivity to the choice of hyperparameters comes from the setup of *EM+D-REX*. While *VIGOR* iteratively improves its policies to match the distribution of the expert demonstrations under the geometric behavioral descriptors, *EM+D-REX* trains a reward function once and then uses this recovered reward function for training policies on the test contexts. As a result, any mistake in the recovered reward will directly influence the way that the newly trained policies are optimized.

Parameter	Default	Value	Component				
			c_0	c_1	c_2	c_3	c_4
Reference			0.0980	0.1593	0.3403	0.5085	0.9838
Architecture	1d-CNN	LSTM	2.3008	3.1438	3.7235	4.1564	4.8256
		MLP	1.9426	2.6433	2.8538	3.1319	3.2751
#Components	5	1	1.0826	–	–	–	–
		3	0.2518	0.9439	1.3415	–	–
#Train Contexts	6	1	1.4140	1.6951	1.8849	1.9518	1.9968
		3	0.5727	0.6553	0.9664	1.4611	1.8949
		12	0.0006	0.1356	0.2014	0.4717	0.6213
#EM Components	3	1	4.9473	5.1427	5.5222	5.7091	5.9643
		2	6.8412	6.8745	6.9075	6.9686	7.0785
#Demonstrations	5	10	4.9109	5.6341	6.0941	6.5230	6.9460
#Layers	2	1	0.8606	1.1513	1.3465	1.7573	2.2349
		3	0.2137	0.3837	0.6393	0.9972	1.5263
#Channels	64	32	0.7734	0.9448	1.3226	1.5071	1.7078
		128	0.1065	0.4228	0.8224	1.6059	2.3021
#Noise Levels	5	2	0.7093	0.9526	1.3025	1.7982	2.5592
		10	0.8883	1.0487	1.2440	1.3916	1.5974
Noise Mult.	5	2	1.0936	1.1909	1.4648	1.8100	2.0065
		10	0.1856	0.3376	0.5531	0.9278	1.2271
Base Noise	0.3	0.1	0.7697	1.1637	1.1975	1.3258	1.5184
		1.0	0.2503	0.4700	0.6243	0.8382	1.0644
		3.0	0.9084	1.1659	1.5097	1.8004	2.5831
Reward Scale	100	1	2.9353	3.0236	3.0985	3.1735	3.2731
		10	0.2810	0.3316	0.3890	0.5140	0.8049
		1000	0.1084	0.2472	0.5330	0.8098	1.1003
		10000	0.1131	0.2744	0.6212	0.8521	1.1223

Table A.1.: Parameter study of *EM+D-REX* on the `Planar Reacher` task. Green and red shading indicates improvement and degradation in performance relative to the reference configuration, respectively. *EM+D-REX* performs well if tuned correctly, but is very sensitive to its choice of hyperparameters.

A.2. Hyperparameters and training details

A.2.1. Environment Parameters

Table A.4 gives an overview of default environment parameters. Parameters with a '*' are varied in the parameter studies. For each environment, we randomly draw both training

Parameter	Default	Value	Component c_0	c_1	c_2	c_3	c_4
Reference			0.0000	0.0003	0.0011	0.0105	0.0322
#Discriminators	5	1	0.0001	0.0003	0.0022	0.0115	0.0403
		3	0.0000	0.0004	0.0013	0.0135	0.0454
Loss	Stepwise BCE	BCE	0.0001	0.0004	0.0044	0.0126	0.0461
#Train Contexts	6	1	0.0846	0.0988	0.1110	0.1349	0.1476
		3	0.0036	0.0058	0.0100	0.0189	0.0712
#Channels	64	32	0.0000	0.0003	0.0014	0.0097	0.0426

Table A.2.: Parameter study of *VIGOR* on the `Panda Reacher` task. Green and red shading indicates improvement and degradation in performance relative to the reference configuration, respectively.

and test contexts at runtime from a fixed set of contexts, making sure that training and test contexts are disjoint for any given seed.

A.2.2. Algorithm Hyperparameters

All neural networks are trained in `PyTorch` (Paszke et al., 2019) using the Adam (Kingma et al., 2015) optimizer. We employ Dropout (Srivastava et al., 2014-01), early stopping and Batch Normalization (Ioffe et al., 2015) to avoid overfitting. All methods are trained until convergence. All mixture policies use 5 components for all experiments.

All methods use the geometric descriptors of the respective environments as input. We use a learning rate of $3.0e{-4}$ for all methods except *EM+D-REX*, for which we found a learning rate of $3.0e{-5}$ to be more stable. *VIGOR* and *EM+D-REX* both use the equations introduced in (Becker et al., 2020) for updating their policies. We found that the method performed very similarly for a range of tested KL-Bound hyperparameters, and thus set it to 0.2 for all experiments for simplicity. We also always draw a sufficient amount of samples to estimate a full-rank quadratic surrogate to update the parameters of their mixture models in each iteration. Both *VIGOR* and *EM+D-REX* also make use of an ensemble with 5 networks, Dropout of 0.2 and use early stopping with a validation split of 0.1. We do not train the categorical distribution for either *VIGOR* or *EM+D-REX*. We find that Batch Normalization improves performance for *EM+D-REX*, but not for *VIGOR*, and as such only use it for *EM+D-REX*. Table A.5 lists the remaining hyperparameters by task for *VIGOR*, Table A.6 those of *EM+D-REX*.

For both BC and BC-GMM we find that training for 3 000 and 30 000 epochs on the state-action and trajectory-based settings works best respectively. Both use an entropy regularization term with a weight of $1.0e{-3}$. For BC-GMM, we additionally add an option to either train or not train the categorical distribution of the mixture to our hyperparameter search. If not trained, the distribution defaults to a uniform distribution over all

Parameter	Default	Value	Component c_0	c_1	c_2	c_3	c_4
Reference			0.0048	0.0049	0.0050	0.0053	0.0055
#Train Contexts	6	1	0.1024	0.1045	0.1050	0.1056	0.1069
		3	0.0423	0.0439	0.0444	0.0449	0.0457
#EM Components	1	2	0.0220	0.0243	0.0252	0.0279	0.0324
		3	0.0747	0.0754	0.0770	0.0785	0.0792
#Channels	64	128	0.0175	0.0181	0.0185	0.0189	0.0193
#Noise Levels	10	5	0.0276	0.0293	0.0299	0.0305	0.0315
		20	0.0178	0.0182	0.0184	0.0188	0.0193
Noise Mult.	10	5	0.0246	0.0266	0.0271	0.0275	0.0332
		20	0.0045	0.0050	0.0054	0.0056	0.0061
Base Noise	0.3	0.1	0.0370	0.0385	0.0391	0.0395	0.0405
		1.0	0.0092	0.0093	0.0096	0.0098	0.0099
		3.0	0.0949	0.0955	0.0968	0.0972	0.0993
Reward Scale	1000	1	0.7115	0.7273	0.7394	0.7522	0.7703
		10	0.1356	0.1453	0.1510	0.1579	0.1730
		100	0.0075	0.0081	0.0089	0.0096	0.0107

Table A.3.: Parameter study of *EM+D-REX* on the `Panda Reacher` task. Green and red shading indicates improvement and degradation in performance relative to the reference configuration, respectively. *EM+D-REX* often collapses to a single solution for the task, which can be seen by the proximity of all 5 components in each bar. The method is again very susceptible to the choice of hyperparameters, which need to be tuned on a by-task basis.

Table A.4.: Default parameters for the `Planar Reacher`, `Panda Reacher` and Box Pusher environments.

Parameter	`Planar Reacher`	`Panda Reacher`	Box Pusher
Trajectory length (T)	30	50	50
ProMP basis functions	5	8	7
Action dimension	5	6	2
Context dimension	4	6	3
Dimension of geometric descriptors	4	4	19
Evaluation samples per component	100	100	5
Number of training contexts ($\lvert\mathbf{c}_{\text{train}}\rvert$)	6	6	6
Training Demonstrations per context	5	5	3
Evaluation samples per component	100	100	5

components, similar to that of *VIGOR* and *EM+D-REX*. Remaining hyperparameters by task for *BC(S)* and BC-GMM(S) are given in Table A.7 and Table A.8 respectively. Those of *BC(T)* and *BC-GMM(T)* are given in Table A.9 and Table A.10. Hyperparameters for *GAIL* are listed in Table A.11.

Table A.5.: Hyperparameters for *VIGOR*. Parameters with a '*' were optimized using grid searches.

Parameter	`Planar Reacher`	`Panda Reacher`	`Box Pusher`
$1d$-CNN layers *	2	3	4
Neurons per layer *	32	64	128
CNN kernel size	5	7	7
Batch size	64	64	64

Table A.6.: Hyperparameters for *EM+D-REX*. Parameters with a '*' were optimized using grid searches.

Parameter	`Planar Reacher`	`Panda Reacher`	`Box Pusher`
$1d$-CNN layers *	2	2	4
Neurons per layer *	64	64	64
CNN kernel size	5	7	7
Batch size	128	128	128
Fit EM Components *	3	1	1
Total EM Samples	8 192	16 384	16 384
Base Noise *	0.3	0.3	0.3
Noise Levels *	5	10	5
Maximum Noise Multiplier *	5	10	2
Reward Scale *	100	1 000	1000

Table A.7.: Hyperparameters for state-action-based Behavioral Cloning (*BC(S)*). Parameters with a '*' were optimized using grid searches.

Parameter	`Planar Reacher`	`Panda Reacher`	`Box Pusher`
MLP layers *	2	2	3
Neurons per layer *	64	32	64
Batch size *	128	64	64
Include target encoding	False	True	–
State-action framestacks *	1	1	1

Table A.8.: Hyperparameters for state-action-based Behavioral Cloning with a Gaussian Mixture Model policy (BC-GMM(S)). Parameters with a '*' were optimized using grid searches.

Parameter	`Planar Reacher`	`Panda Reacher`	`Box Pusher`
MLP layers *	2	2	3
Neurons per layer *	64	64	64
Batch size *	64	64	64
Include target encoding	False	True	–
State-action framestacks *	5	1	1
Train categorical distribution *	True	True	False

Table A.9.: Hyperparameters for trajectory-based Behavioral Cloning (*BC(T)*). Parameters with a '*' were optimized using grid searches.

Parameter	`Planar Reacher`	`Panda Reacher`	`Box Pusher`
MLP layers *	4	4	3
Neurons per layer *	256	64	128
Batch size *	4	4	4

Table A.10.: Hyperparameters for trajectory-based Behavioral Cloning with a Gaussian Mixture Model policy (*BC-GMM(T)*). Parameters with a '*' were optimized using grid searches.

Parameter	`Planar Reacher`	`Panda Reacher`	`Box Pusher`
MLP layers *	4	3	3
Neurons per layer *	64	64	128
Batch size	4	4	4
Train categorical distribution *	False	False	False

Table A.11.: Hyperparameters for *GAIL*. Parameters with a '*' were optimized using grid searches.

Parameter	`Planar Reacher`	`Panda Reacher`	`Box Pusher`
Discriminator MLP layers *	2	2	3
Discriminator neurons per layer *	32	64	64
Discriminator batch size *	128	64	64
Policy MLP layers	2	2	2
Policy MLP neurons per layer	32	32	32
Policy batch size *	128	64	256
Total time-steps *	$1e6$	$3e6$	$1e6$
Include target encoding	True	False	–
State-action framestacks *	5	5	1
Share networks	False	False	False
Policy steps	2 048	2 048	2 048

B. Appendix for Chapter 4

B.1. Systems of Equations

The following briefly describes the system of equations used for our experiments, including weak PDE formulations, domains, initial and boundary conditions, as introduced in Section 2.3.1. Each environment uses a fixed PDE formulation and varies either the process conditions, the used domain, or both. For some environments, we add additional information as node features to the observation graph of the MPN policy.

B.1.1. `Laplace`

Consider a domain Ω bounded internally by $\partial\Omega_{\text{in}}$ and externally by $\partial\Omega_{\text{out}}$. We seek a function $u(x)$ satisfying `Laplace`'s Equation

$$\int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x = 0 \quad \forall v \in V,$$

where $v(x)$ denotes the test function and the solution $u(x)$ has to satisfy the Dirichlet boundary conditions

$$u(x) = 0, x \in \partial\Omega_{\text{in}} \quad \text{and} \quad u(x) = 1, x \in \partial\Omega_{\text{out}}.$$

The domain is a unit square $(0,1)^2$ with a squared hole defined by the inner boundary $\partial\Omega_{\text{in}}$. For each system of equations, the hole's size is uniformly sampled from $\mathcal{U}(0.05, 0.25)$ in both directions, while its center is placed randomly in $\mathcal{U}(0.2, 0.8)^2$. For a given mesh, the minimum distance to $\partial\Omega_{\text{in}}$ of each face midpoint is included as a node feature.

B.1.2. `Poisson`

We address `Poisson`'s Equation

$$\int_\Omega \nabla u \cdot \nabla v \mathrm{d}x = \int_\Omega f v \mathrm{d}x \quad \forall v \in V,$$

where $f(x)$ is the load function, and $v(x)$ is the test function. The solution $u(x)$ must satisfy $u(x) = 0$ on $\partial\Omega$. We use L-shaped domain Ω, defined as $(0,1)^2 \setminus (p_0 \times (1,1))$, where the lower left corner p_0 is sampled from $\mathcal{U}(0.2, 0.95)$ in x and y direction.

We employ a Gaussian Mixture Model (GMM) with three components for each problem's load function. The mean of each component is drawn from $\mathcal{U}(0.1, 0.9)^2$, with rejection sampling ensuring all means reside within Ω. We then independently draw diagonal covariances from a log-uniform distribution $\exp(\mathcal{U}(\log(0.0001, 0.001)))$ and randomly rotate them to yield a full covariance matrix. Component weights are generated from $\exp(N(0, 1)) + 1$ and subsequently normalized. The load function evaluation at each face's midpoint is used as a node feature.

B.1.3. Stokes Flow

Let $u(x)$ and $p(x)$ represent the velocity and pressure fields, respectively, in a channel flow. We analyze a `Stokes Flow`, aiming to solve for u and p without a forcing term, i.e.,

$$\nu \int_\Omega \nabla v \cdot \nabla u \, \mathrm{d}x - \int_\Omega (\nabla \cdot v) p \, \mathrm{d}x = 0 \quad \forall v \in V,$$

$$\int_\Omega (\nabla \cdot u) q \, \mathrm{d}x = 0 \quad \forall q \in V,$$

with test functions $v(x)$ and $q(x)$ (Quarteroni et al., 2009). The velocity field at the inlet is defined as

$$u(x = 0, y) = u_\mathrm{P} y(1 - y) + \sin(\varphi + 2\pi y).$$

At the outlet, we impose $\nabla u(x = 1, y) = 0$, and assume a no-slip condition $u = 0$ at all other boundaries. For numerical stability, we utilize P_2/P_1 Taylor-Hood elements, with quadratic velocity and linear pressure shape functions (John et al., 2016).

The inlet profile parameter u_P is sampled from a log-uniform distribution $\exp(\mathcal{U}(\log(0.5, 2)))$. The domain is structured as a unit square with three rhomboid holes of length 0.4 and height 0.2. The holes are centered at $y \in \{0.2, 0.5, 0.8\}$ and we randomly sample their x-coordinate from $\mathcal{U}(0.3, 0.7)$. We optimize the meshes with respect to the velocity vector and present a scalar error as the norm of the vectorized velocity error.

B.1.4. Linear Elasticity

We investigate the steady-state deformation of a solid under stress caused by displacements at the boundary $\partial\Omega$ of the domain Ω. Let $u(x)$ be the displacement field, $v(x)$ the test function, and

$$\varepsilon(u) = \frac{1}{2}(\nabla u + (\nabla u)^\top)$$

the strain tensor. The linear-elastic and isotropic stress tensor is given as

$$\sigma(\varepsilon) = 2\mu\varepsilon + \lambda \mathrm{tr}(\varepsilon) I,$$

using Lamé parameters

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \quad \text{and} \quad \mu = \frac{E}{2(1+\nu)}$$

with a problem specific Young's modulus $E = 1$ and Poisson ratio $\nu = 0.3$. Without body forces, the problem is given as (Zienkiewicz et al., 2005)

$$\int_\Omega \sigma\left(\varepsilon\left(u\right)\right) : \varepsilon\left(v\right)\, \mathrm{d}x = 0 \quad \forall v \in V.$$

We use the same class of L-shaped domains as in `Poisson` in Section B.1.2. We fix the displacement of the left boundary to 0, i.e., $u(x = 0, y) = 0$, and randomly sample a displacement direction from $\mathcal{U}(0, \pi)$ and magnitude from $\mathcal{U}(0.2, 0.8)$ to displace the right boundary as $u(x = 1, y) = u_P$. The stress $\sigma \cdot n = 0$ is zero normal to the boundary at both the top and bottom of the part. The problem-dependent displacement vector u_P is added as a globally shared node feature to all elements. Our objective in the resulting `Linear Elasticity` environment includes the norm of the displacement field u and the resulting Von-Mises stress, using an equal weighting of 0.5 between the two.

B.1.5. `Heat Diffusion`

Using temperature $u(x)$, a test function $v(x)$, and a thermal diffusivity constant $a = 0.001$, we address a non-stationary `Heat Diffusion` problem defined by

$$\int_\Omega \frac{\partial u}{\partial t}\,\mathrm{d}x + \int_\Omega a\nabla u \cdot \nabla v\,\mathrm{d}x = \int_\Omega fv\,\mathrm{d}x \quad \forall v \in V.$$

Here, $f(x, y)$ is a position-dependent heat distribution

$$f(x, y) = 1\,000 \exp\left(-100\left(\left(x - x_p(\tau)\right) + \left(y - y_p(\tau)\right)\right)\right).$$

The position of the heat source's path $p_\tau(\tau) = \left(x_p(\tau), y_p(\tau)\right)$ is linearly interpolated over time as

$$p_\tau = p_0 + \frac{\tau}{\tau_{\max}}(p_{\tau_{\max}} - p_0).$$

The start and goal positions p_0 and $p_{\tau_{\max}}$ are randomly drawn from the entire domain. The temperature $u \in \partial\Omega$ is set to zero on all boundaries. We use a total of $\tau_{\max} = 20$ uniform time steps in $\{0.5, \dots, 10\}$ and an implicit Euler method for the time-integration. Domain geometry is derived from 10 equidistant points on a circle centered at $(0.5, 0.5)$ with a radius of 0.4. Each point is randomly distorted by a value $\mathcal{U}(-0.2, 0.2)^2$ before we proceed to calculate the points convex hull as our domain. We finally normalize the convex hull to lie in $(0, 1)^2$. The result is a family of convex polygons with up to 10 vertices. We provide and measure the error and solution of the final simulation step, and provide the distance to the start and end position of the heat source as additional node features for each element.

B.1.6. `Neumann Boundaries`

We adapt `Poisson` in Section B.1.2 to include `Neumann Boundary` conditions. Using star-shaped domains, we randomly select half the line segments $\partial\Omega_l$ that make up the boundary of the star as Neumann boundaries, and use zero Dirichlet boundary conditions for the remaining segments. Each selected line segment is assigned a random sinusoidal Neumann boundary condition

$$\nabla u \cdot n = 10 \cdot A \cdot p_x \cdot (1 - p_x) \cdot \sin(\pi \nu p_x), \quad x \in \partial\Omega_l,$$

where $n(x)$ denotes the normal vector of the boundary segment, $p_x \in [0, 1]$ is the normalized position along the line segment, $A \in \mathcal{U}(1, 3)$ the amplitude and $\nu \in \{3, 5, 7\}$ the frequency.

To create the star-shaped domains, we define an outer radius of 0.5 and an inner radius of 0.2 from a point centered at $(0.5, 0.5)$. We then sample a number of star points in $\mathcal{U}(3, 5)$ and equidistantly interleave them between the two boundaries. We randomly distort each point by $\mathcal{U}(-0.05, 0.05)^2$ before applying a random rotation and normalizing the full domain to be within $(0, 1)^2$. Initial meshes are created with a target volume of 0.02 instead of the 0.05 used for the other environments to compensate for the lower total volume of the star-shaped domains, and the diagonal covariances for the GMM load function are drawn from $\mathcal{U}(0.00005, 0.0005)$. Each element gets the distance to the closest Neumann and Dirichlet boundaries as additional node features.

B.1.7. `3D Poisson`

Finally, we extend `Poisson` in Section B.1.2 to a `3D Poisson` environment. We use a rectangular plate of length $l_x = 1$, width $l_y = 0.5$ and height $l_z = 0.1$ as our domain, and create an initial mesh from by dividing it into 0.1^3 cubes and triangulating each cube. We assign Dirichlet boundary conditions

$$u(x = 0, y, z) = u(x = 1, y, z) = u(x, y = 0, z) = u(x = 1, y = 0.6, z) = 0$$

on the sides of the plate and define natural boundary conditions on the lower $z = 0$ and upper side $z = 0.1$ of the plate. We use tetrahedral elements for this environment, allowing for a fully volumetric mesh, and solve with an iterative instead of a direct solver to accommodate the larger number of elements. The remesher uses longest edge bisection (Rivara, 1984; Suárez et al., 2005) to halve marked elements, whereas the 2-dimensional domains employ the red-green-blue refinement method (Carstensen, 2004).

B.2. Mean and Maximum Error Metrics

In addition to the squared error evaluated in Section 4.4, we present results for a mean and maximum mesh error metric. We approximate the maximum error as the average of

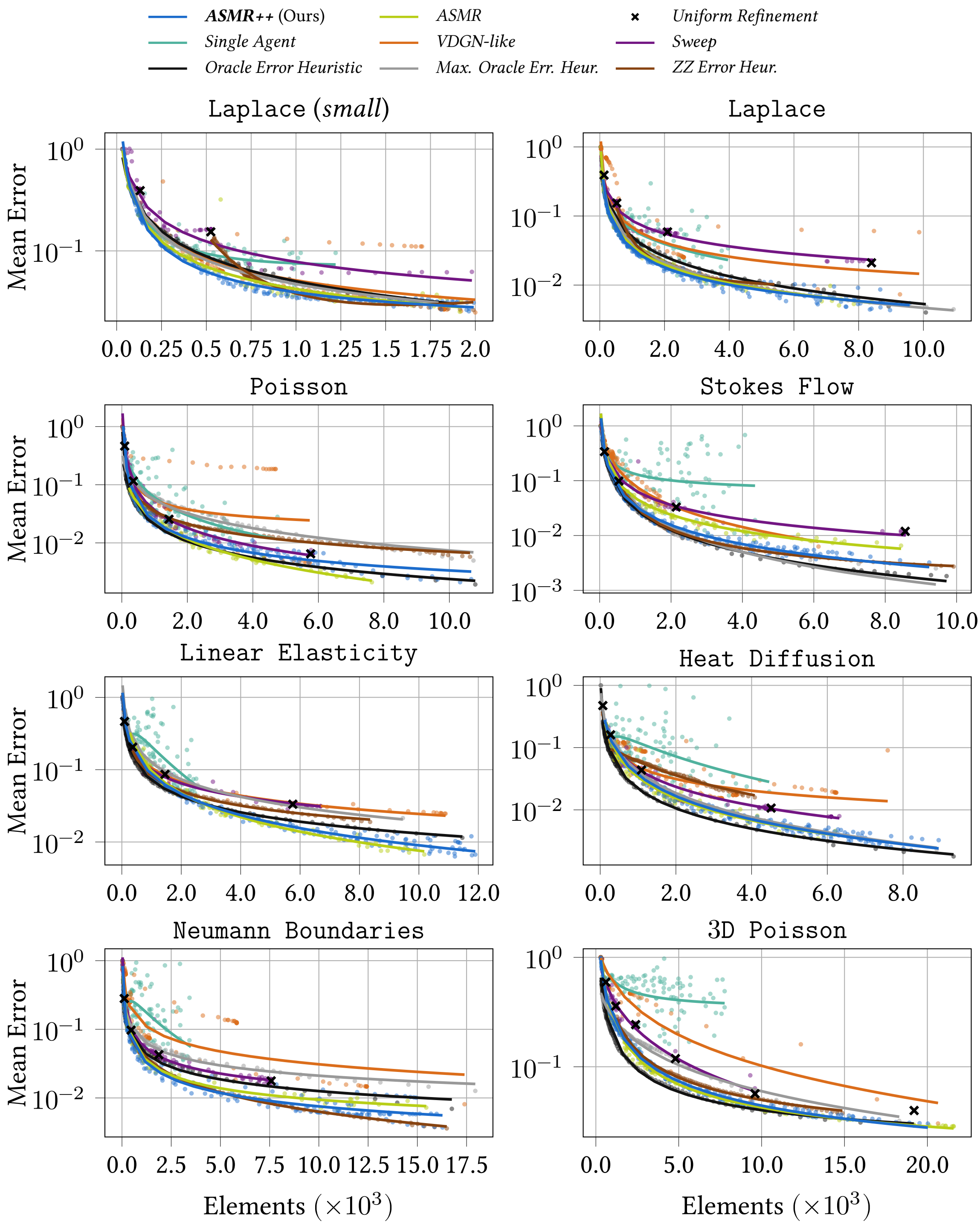


Figure B.1.: Pareto plot of normalized mean errors and number of final mesh elements for all environments. *ASMR++* performs on par with or better than *ASMR* on all environments, and both methods significantly outperform all RL-AMR baselines.

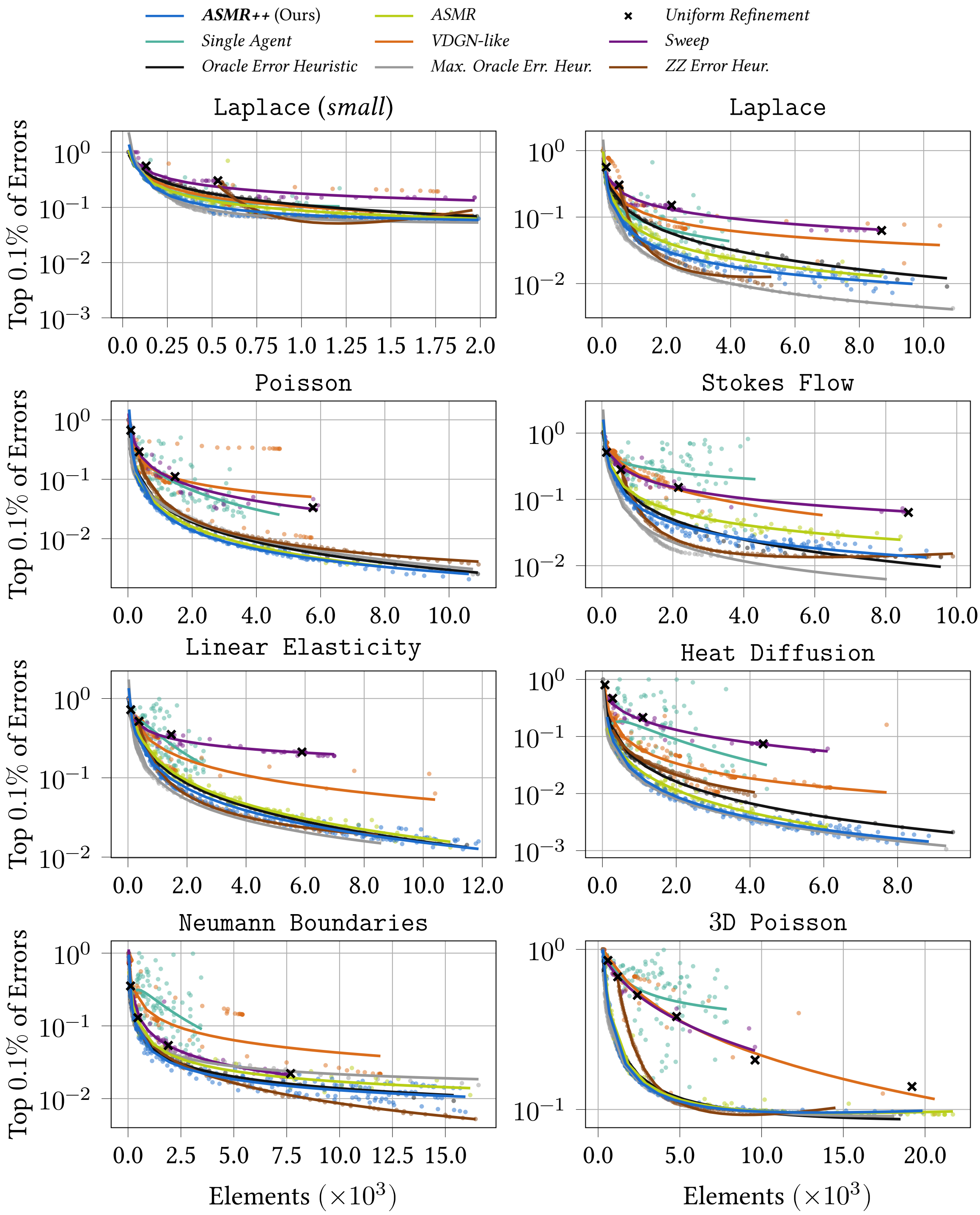


Figure B.2.: Pareto plot of normalized maximum element error and number of final mesh elements for all environments. *ASMR++* directly minimizes the error of the maximum element during training. In contrast, *ASMR* optimizes a scaled variant of the average error. Consequently, *ASMR++* surpasses *ASMR* on all environments. Both methods outperform all other RL-AMR baselines.

the Top $0.1\,\%$ of errors of all integration points $\mathbf{p}(M_m^*)$ to make it robust to outliers. We normalize both metrics by the error of the initial mesh for each system of equations.

Figure B.1 shows pareto plots for a normalized mean mesh error for all environments. The *Oracle Error Heuristic* outperforms its maximum error counterpart, likely because it selects elements with a high integrated error rather than elements with a high maximum error for refinement, thus targeting areas with a high mean error. *ASMR++* performs on par with or better than *ASMR* on average, even though it explicitly optimizes a decrease in maximum mesh error rather than a decrease in its mean error, likely due to its local optimization objective. Since the mean mesh error is less sensitive to outliers, more uniform refinements such as those produced by *Sweep* perform better than on other metrics. The mean error directly quantifies the difference between the solution calculated on the fine ground truth mesh and that produced by the different methods. For most environments, *ASMR++* produces meshes that reduce the error between the initial mesh and the ground truth by more than $99\,\%$ with only a few thousand elements.

Figure B.2 visualizes pareto plots for all environments for the Top $0.1\,\%$ mesh error. Here, the *Maximum Oracle Error Heuristic* excels when compared to the regular *Oracle Error Heuristic*, likely because it refines elements that have the highest maximum error. Similarly, *ASMR++* outperforms *ASMR* on all environments, as it directly minimizes the maximum error instead of a scaled version of the average mesh error. Methods that produce relatively uniform meshes, such as *Sweep*, perform considerable worse on this approximated maximum error than on the other metrics, likely because uniform refinements produce a lot of elements on mesh areas that do not participate in the maximum mesh error. Notably, the maximum error for the `3D Poisson`'s equation is bound at around 0.1. This is likely the case because we use longest edge bisection for the adaptive refinements, which produces refined meshes with local elements that significantly differ from that of the fine-grained uniform ground truth reference. As bounding the maximum prediction error is important in many applications, these uniform meshes can potentially waste a lot of computational resources or lead to worse results than the highly adaptive meshes produced by, e.g. *ASMR++*. Overall, both metrics are consistent with the results in Figure 4.8 and Figure 4.9.

B.3. Further Parameter Studies and Ablations

B.3.1. Initial Meshes for the *ZZ Error Heuristic*

Figure B.3 demonstrates the performance of the *ZZ Error Heuristic* applied to the initial mesh, in comparison to its performance when the process begins with either one or two uniform mesh refinements. The results indicate that initial uniform refinements significantly improve the heuristic's effectiveness, likely because the uniform refinements make it easier to identify gradients in key areas, which may be overlooked if the mesh is too coarse. Consequently we opt for the twice-refined mesh approach for all experiments.

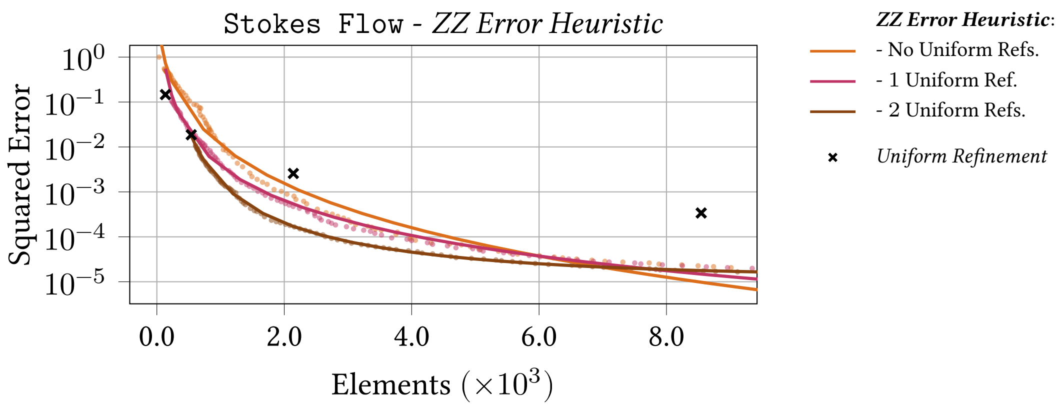


Figure B.3.: Pareto plot of normalized squared errors and number of final mesh elements on `Stokes Flow` for the Zienkiewicz-Zhu Error Estimator (ZZ Error) *Heuristic* for 0, 1 and 2 initial uniform mesh refinements. A finer initial mesh leads to improved performance for the *ZZ Error Heuristic* but prevents the creation of very coarse meshes, indicating that the initial element size should be tuned for the *ZZ Error Heuristic*.

In comparison, the RL-AMR methods do not need to tune their initial mesh, as the methods are trained to produce optimal refinement sequences.

B.3.2. Element Penalty

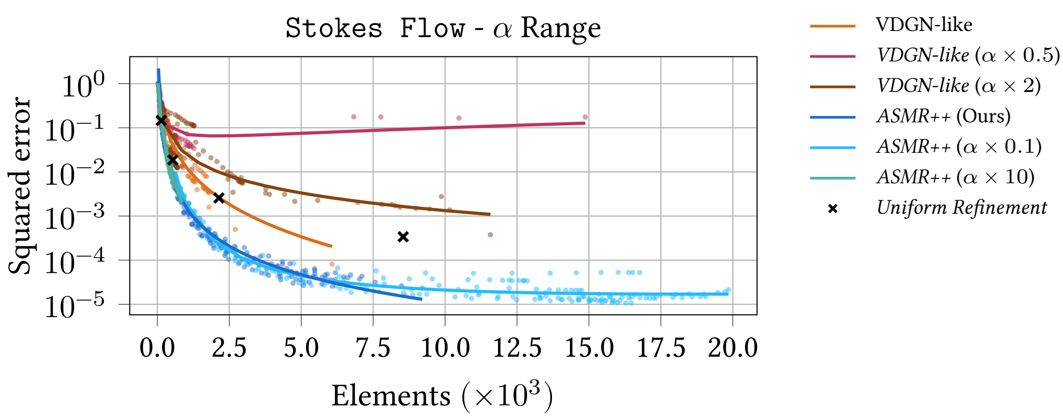


Figure B.4.: Pareto plot of normalized squared errors and number of final mesh elements on `Stokes Flow` for different ranges of the element penalty α for *ASMR++* and *VDGN-like*. Increasing or decreasing the element penalty of *ASMR++* by a factor of 10 leads to finer meshes for larger penalties and vice versa. Regardless of the scale of the element penalty, *ASMR++* produces high-quality meshes. The *VDGN-like* baseline is comparatively unstable, yielding worse results when adapting the element penalty.

Figure B.4 shows the effect of different ranges for the element penalty α for *ASMR++* and *VDGN-like*. Here, we scale the minimum and maximum sampled penalty by a constant factor during training and inference. While *VDGN-like* works well for the chosen penalty, it is generally unstable and fails to produce consistent refinements for finer meshes,

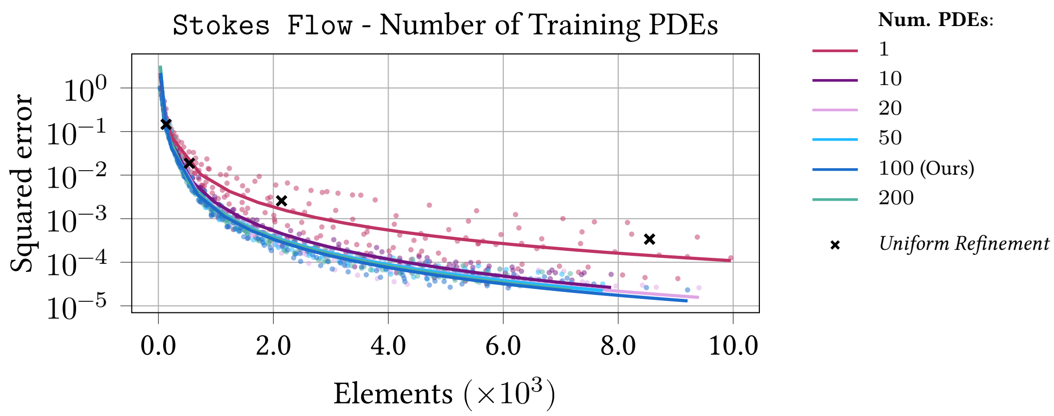


Figure B.5.: Pareto plot of normalized squared errors and number of final mesh elements on `Stokes Flow` for different numbers of training systems of equations. *ASMR++* performs better than uniform on novel evaluation settings when trained on a single system of equations, and improves for up to 100 systems of equations in the training set.

regardless of whether the range of α is scaled by a factor of 2. In comparison, *ASMR++* produces high-quality meshes for different element penalty ranges. When decreasing all penalties by a factor of 10, *ASMR++* produces finer meshes with significantly more elements compared to the regular penalty. Similarly, increasing the penalty makes refinements less attractive, reducing the number of produced elements.

B.3.3. Number of Training Systems of Equations

We generally train *ASMR++* on 100 systems of equations, each of which consists of a randomly sampled domain and random process conditions. Figure B.5 explores how fewer or more systems of equations during training affect performance, finding that performance improves for up to 100 points of data. Beyond 100 training systems of equations, performance gains diminish. During training, the reward function requires solving a fine-grained reference mesh M^* for each system of equations, which is computationally expensive. We thus use 100 training systems of equations for our main experiments. Albeit noisy and inconsistent, *ASMR++* performs better than uniform refinements on the unseen evaluation set when trained on a single system of equations, likely due to the mostly local optimization objective and network architecture.

B.3.4. Network Architecture

We ablate our choice of network architecture on the right side of Figure B.6. Using Graph Attention Networks (Velickovic et al., 2018) with edge features instead of our Message Passing Network (MPN) does not significantly impact performance. *ASMR++* uses Edge Dropout (Rong et al., 2019) of 0.1, i.e., it randomly removes every tenth edge of the

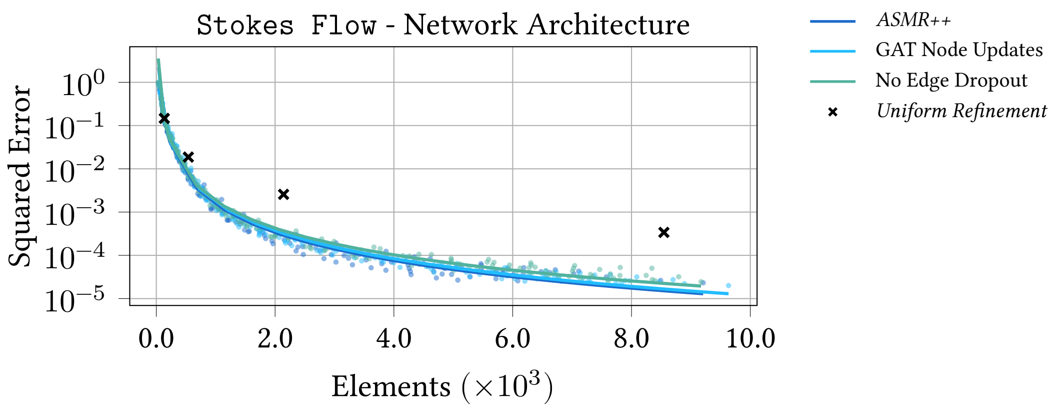


Figure B.6.: Pareto plot of normalized squared errors and number of final mesh elements on `Stokes Flow` for different network architectures. Performing message passing updates with GATs with edge features instead of MPNs does not significantly impact performance. Omitting Edge Dropout leads to slightly worse refinements.

observation graph, regularizing the training data and incentivizing the model to learn general patterns instead of spurious correlations. Omitting this dropout leads to slightly worse refinements.

B.4. Hyperparameters

B.4.1. General Hyperparameters

The following section lists important hyperparameters used for the training of all methods. Hyperparameters are kept consistent across all environments and RL-AMR methods unless mentioned otherwise.

Neural Networks. We implement all neural networks in `PyTorch` (Paszke et al., 2019) and optimize them using ADAM (Kingma et al., 2015) with a learning rate of $3.0e{-}4$ and batch size of 32 unless mentioned otherwise. All MLPs use 2 hidden layers and a latent dimension of 64. Each MPN consists of 2 message passing steps, where each update function is represented as an MLP with LeakyReLU activation functions. We apply Layer Normalization (Ba et al., 2016) and Residual Connections (He et al., 2016) independently after each node and edge feature update, and use Edge Dropout (Rong et al., 2019) of 0.1 during training. The edge feature aggregations $\bigoplus$ are mean aggregations. We use separate architectures for the policy and the value function, i.e., we do not share weights between the policy and value function. The policy and value function heads are MLPs with *tanh* activation functions acting on the final latent node features of the MPN.

PPO. For PPO, we follow previous work (Andrychowicz et al., 2021) to select important hyperparameters and code-level optimizations. We train each PPO policy for a total of 400 iterations, except for `3D Poisson`, where we only use 200 iterations. In each iteration, the algorithm samples 256 environment transitions and performs 5 epochs of optimization. The value function loss is multiplied with a factor of 0.5. We clip the gradient norm to 0.5 and choose clip ranges of 0.2 for the policy and value function. We normalize the observations with a running mean and standard deviation. Advantages are estimated via Generalized Advantage Estimate (Schulman et al., 2015b) using $\lambda = 0.95$. We compute an agent's advantage by subtracting the agent-wise value estimates from the return in Equation 4.8.

DQN. For DQN-based approaches, we draw 500 initial samples with a random policy for the replay buffer and then train for $24 \cdot 400 = 9600$ steps, where each step consists of executing and storing an environment transition and then drawing a random mini-batch with replacement from the buffer for a single gradient update. We experimented with more training steps in preliminary experiments, finding that they do no significantly improve performance or stabilize training, but may lead to increased memory footprint and longer runtimes. We update the target networks using Polyak averaging at a rate of 0.99 per step. We select training actions using a Boltzmann distribution over the predicted Q-values per agent, where we linearly decrease the temperature of the distribution from 1 to 0.01 in the first half of training. This action selection strategy favors more correlated actions when compared to an epsilon greedy action sampling, which empirically stabilizes the training for our problem setting of iterative mesh refinement. Further, we follow previous work (Hessel et al., 2018) and combine a number of common improvements for DQNs, namely double Q-learning (Van Hasselt et al., 2016), dueling Q-networks (Wang et al., 2016) and prioritized experience replay (Schaul et al., 2016).

B.4.2. Baseline-Specific Parameters

ASMR. We use the reward of Equation 4.7, an infinite horizon RL setting with $\gamma = 0.99$ and omit the edge dropout in the observation graph for *ASMR*. We additionally use an agent mapping without the normalization factor $|M^t|/|M^{t+1}|$ in Equation 4.4.

Single Agent. We use a maximum refinement depth of 10 refinements per element to avoid numerical instabilities during simulation, skipping actions that try to refine elements that have been refined too often. We consider environment sequences of up to $T = 400$ steps since the method marks only one element at a time, but train a separate policy for every value of T.

Sweep. The *Sweep* baseline moves a single agent to a random mesh element after each training step. The agent then decides if this element should be refined or not. We follow the proposed hyperparameters and have training rollouts of 200 steps (Foucart et al., 2023). Since *Sweep* uses a local agent living on a single mesh element, we use an MLP instead of the MPN for both the policy and value function. Correspondingly, we adapt the input features, using our regular node features and the global resource budget proposed by the authors. We additionally add aggregated neighborhood information in the form of a mean solution and area of the element's neighbors and the average distance to them. The global budget is controlled via a maximum number of elements N_{max}, allowing to train policies that produce refinements of different granularity. We increase the number of environment transitions per PPO step to 512, and the number of DQN steps to $96 * 400 = 38400$. This change is intended to compensate for decreased number of refined elements of each environment step while roughly equating for the computation time of the other methods.

VDGN-like. We use a learning rate of $1.0e{-}5$ instead of $3.0e{-}4$ for the DQN variant of *VDGN-like* to stabilize its training.

B.4.3. Mesh Resolution Parameters

All AMR methods in this work allow for some control over the desired refinement level of the final mesh. *ASMR++*, *ASMR* and *VDGN-like* use an element penalty α that trades off the cost of adding new elements with the benefit of a refinement. *Sweep* considers an element budget N_{max}, attempting to minimize the error of the mesh while staying within this budget. *Single Agent* varies the number of rollout steps T, refining once for every step.

Both *ASMR++* and *VDGN-like* train on a range of α values and condition the policy on it, allowing for adaptive mesh resolutions during inference. Here, we evaluate 20 penalties that are log-uniformly sampled over the training penalty range to cover different final mesh resolutions. As the other methods do not directly support an adaptive penalty during inference, we train 10 separate policies with 10 repetitions each for different refinement parameters. The *Oracle*, *Maximum Oracle*, and *ZZ Error Heuristics* refine based on estimated errors per mesh element. Here, we evaluate 100 values for the error threshold ϵ, which specifies which elements to refine based on the ratio between their error and the maximum element error. Tables Table B.1 and Table B.2 list the minimum and maximum mesh resolution parameters for all environments and RL-AMR methods and heuristics, respectively.

Parameter	***ASMR++*** α	***ASMR*** α	***VDGN-like*** α	***Sweep*** N_{max}	***Single Agent*** T
`Laplace` *(Small)*	$[0.001, 0.1]$	$[0.01, 0.3]$	$[1e{-}5, 5e{-}2]$	$[50, 1000]$	$[5, 100]$
`Laplace`	$[0.001, 0.03]$	$[0.01, 0.5]$	$[1e{-}5, 1e{-}2]$	$[200, 3000]$	$[25, 400]$
`Poisson`	$[0.0001, 0.03]$	$[0.002, 0.1]$	$[2e{-}5, 5e{-}2]$	$[200, 3000]$	$[25, 400]$
`Stokes Flow`	$[0.0005, 0.05]$	$[0.015, 0.3]$	$[3e{-}4, 2e{-}2]$	$[150, 2500]$	$[25, 400]$
`Linear Elasticity`	$[0.00015, 0.03]$	$[0.01, 0.15]$	$[1e{-}5, 1e{-}2]$	$[500, 6000]$	$[25, 400]$
`Heat Diffusion`	$[0.00005, 0.01]$	$[0.003, 0.3]$	$[1e{-}5, 1e{-}2]$	$[400, 5000]$	$[25, 400]$
`Neumann Boundaries`	$[0.00005, 0.03]$	$[0.0075, 0.5]$	$[1e{-}5, 1e{-}1]$	$[200, 3000]$	$[25, 400]$
`3D Poisson`	$[0.0001, 0.1]$	$[0.1, 20.0]$	$[1e{-}5, 1e{-}1]$	$[500, 5000]$	$[25, 400]$

Table B.1.: Ranges for the different refinement hyperparameters for all environments and RL-AMR methods. *ASMR++*, *ASMR* and *VDGN-like* apply a penalty α for each added element. *Sweep* uses an element budget N_{max}. *Single Agent* varies the number of rollout steps T.

Parameter	*Oracle Error* ϵ	*Max. Oracle Err.* ϵ	*ZZ Error* ϵ
`Laplace` (*small*)	$[0.22, 1.0]$	$[0.16, 1.0]$	$[0.002, 1.0]$
`Laplace`	$[0.20, 1.0]$	$[0.12, 1.0]$	$[0.001, 1.0]$
`Poisson`	$[0.12, 1.0]$	$[0.15, 1.0]$	$[0.002, 1.0]$
`Stokes Flow`	$[0.12, 1.0]$	$[0.04, 1.0]$	$[0.001, 1.0]$
`Linear Elasticity`	$[0.06, 1.0]$	$[0.02, 1.0]$	$[0.001, 1.0]$
`Heat Diffusion`	$[0.04, 1.0]$	$[0.02, 1.0]$	$[0.001, 1.0]$
`Neumann Boundaries`	$[0.14, 1.0]$	$[0.24, 1.0]$	$[0.001, 1.0]$
`3D Poisson`	$[0.02, 1.0]$	$[0.02, 1.0]$	$[0.001, 1.0]$

Table B.2.: Ranges for the threshold ϵ of the error-based refinement strategy of the heuristic baselines. All ranges are chosen to facilitate direct comparison to the RL-AMR methods in the quantitative evaluation, i.e., to produce meshes with a comparable number of elements.

B.5. Baseline Visualizations

B.5.1. RL-AMR visualizations

We visualize the final meshes and corresponding FEM solutions of all RL-AMR methods on `Stokes Flow` for 6 different refinement granularities on the same randomly selected system of equations. For *ASMR++* and *VDGN-like*, which condition the policy on the element penalty parameter, we use the same trained model for the 6 mesh refinement granularities. Figure B.7 visualizes the results.

We find that *ASMR++* provides accurate refinements for different values of α using a single policy. The produced meshes are of higher quality than those created by *ASMR*, especially near the corners of the rhomboid obstacles in the domain. All other RL-AMR methods are unable to produce consistent meshes. *Single Agent* tends to over-refine uninteresting regions, likely due to its iterative refinement procedure. In contrast, *Sweep* often collapses to uniform or mostly uniform refinements, which may be a by-product of the difference in training on individual elements and sweeping over all elements at once during inference. The *VDGN-like* baseline produces high-quality refinements in some cases, but is inconsistent across seeds and in some cases fails to optimize for different element penalties with the same model, which could be because its value decomposition objective may scale poorly to large numbers of agents. These qualitative findings are consistent with the error measurements of Figure 4.9.

B.5.2. Heuristic visualizations

We visualize exemplary refinements for the error estimation-based threshold *Heuristics* in Figure B.8. All *Heuristics* greedily refine the elements with their respective largest error estimates, regardless of the resulting decrease in error. This behavior is fully local, which

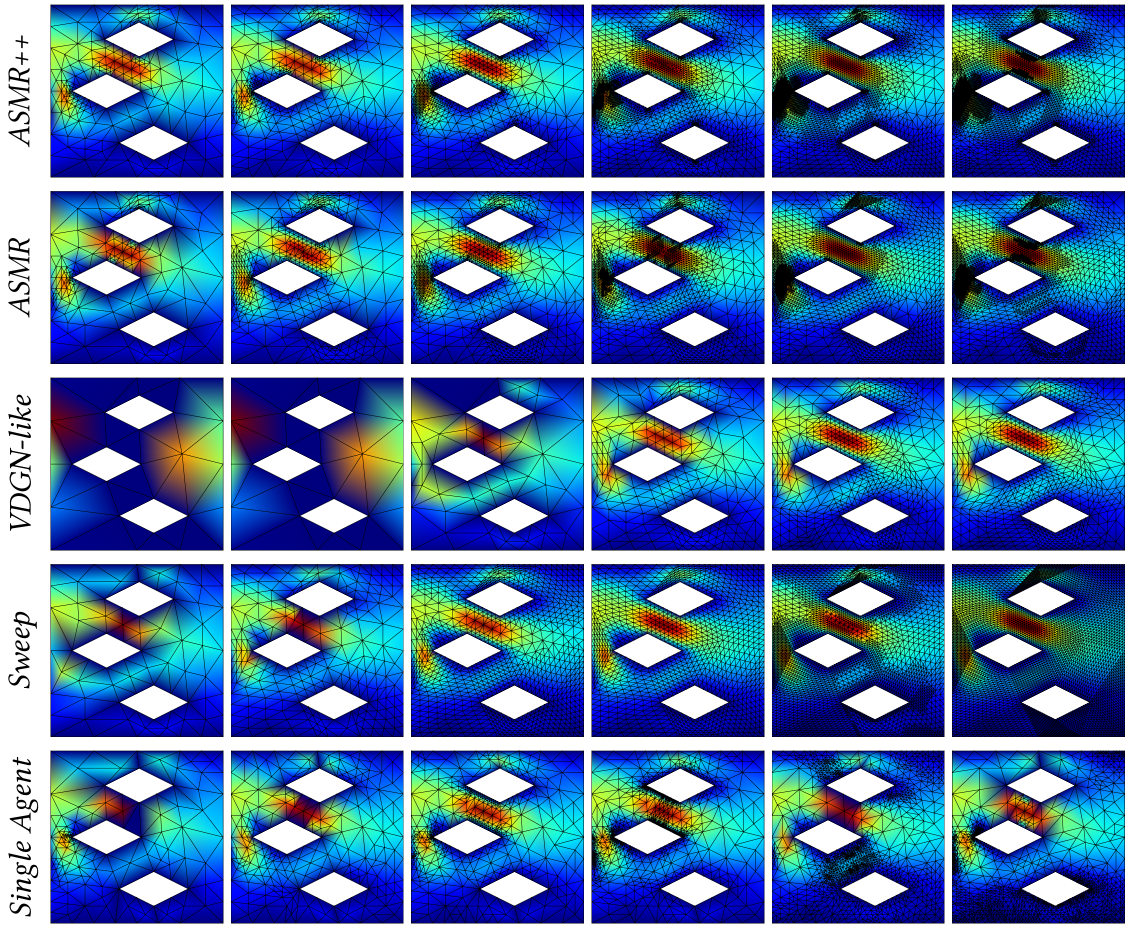


Figure B.7.: Comparison of the different RL-AMR methods for the same `Stokes Flow` system of equations for different target mesh resolutions. The target mesh resolution increases from the left to the right for each row. *ASMR++* and *VDGN-like* use the same model queried with different α values for each row. The other methods use a different policy for each produced mesh.

may cause issues for global dependencies (Strauss, 2007) and conforming refinements. The *ZZ Error Heuristic* uses a smoother error estimate than the oracle heuristics, which leads to potentially sub-optimal but generally more consistent refinements.

B.6. Additional *ASMR* Visualizations

We visualize meshes created by *ASMR++* policies for all considered environments. For the 2-dimensional environments, we follow Section B.5 and use a fixed policy to create refinements with 6 different granularities. We use a random but fixed system of equations for each environment, and only vary the element penalty α. For `3D Poisson`, we instead visualize the same random system of equations and α value for 6 different camera angles. Figure B.9 displays the final refined meshes, as well as the PDE solutions on the meshes.

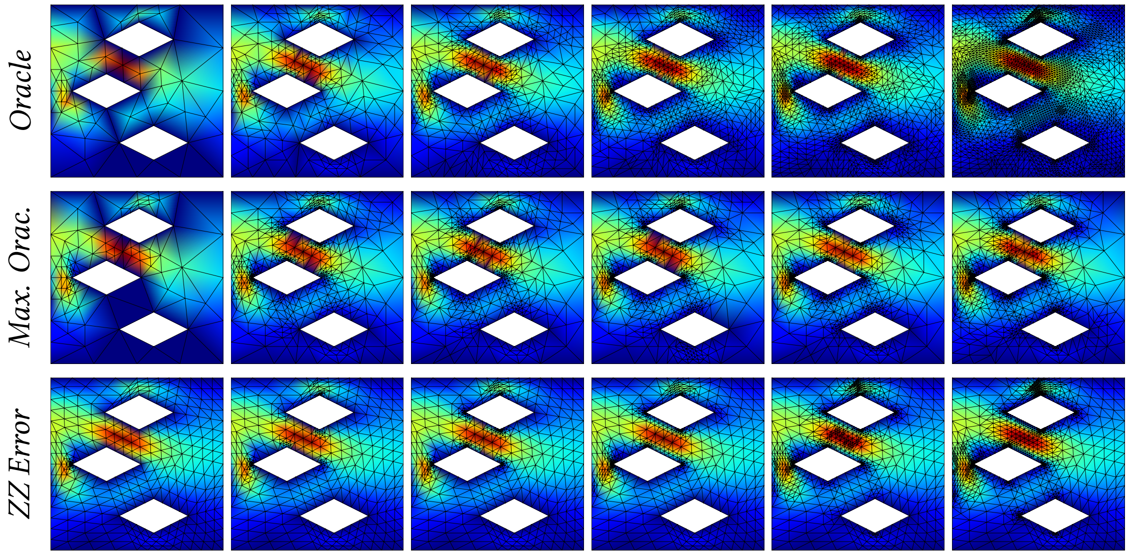


Figure B.8.: Comparison of the different *Heuristic* methods for the same `Stokes Flow` system of equations and different error thresholds ϵ. The target mesh resolution increases from the left to the right for each row.

On all environments, *ASMR++* provides highly accurate refinements across target mesh refinement levels.

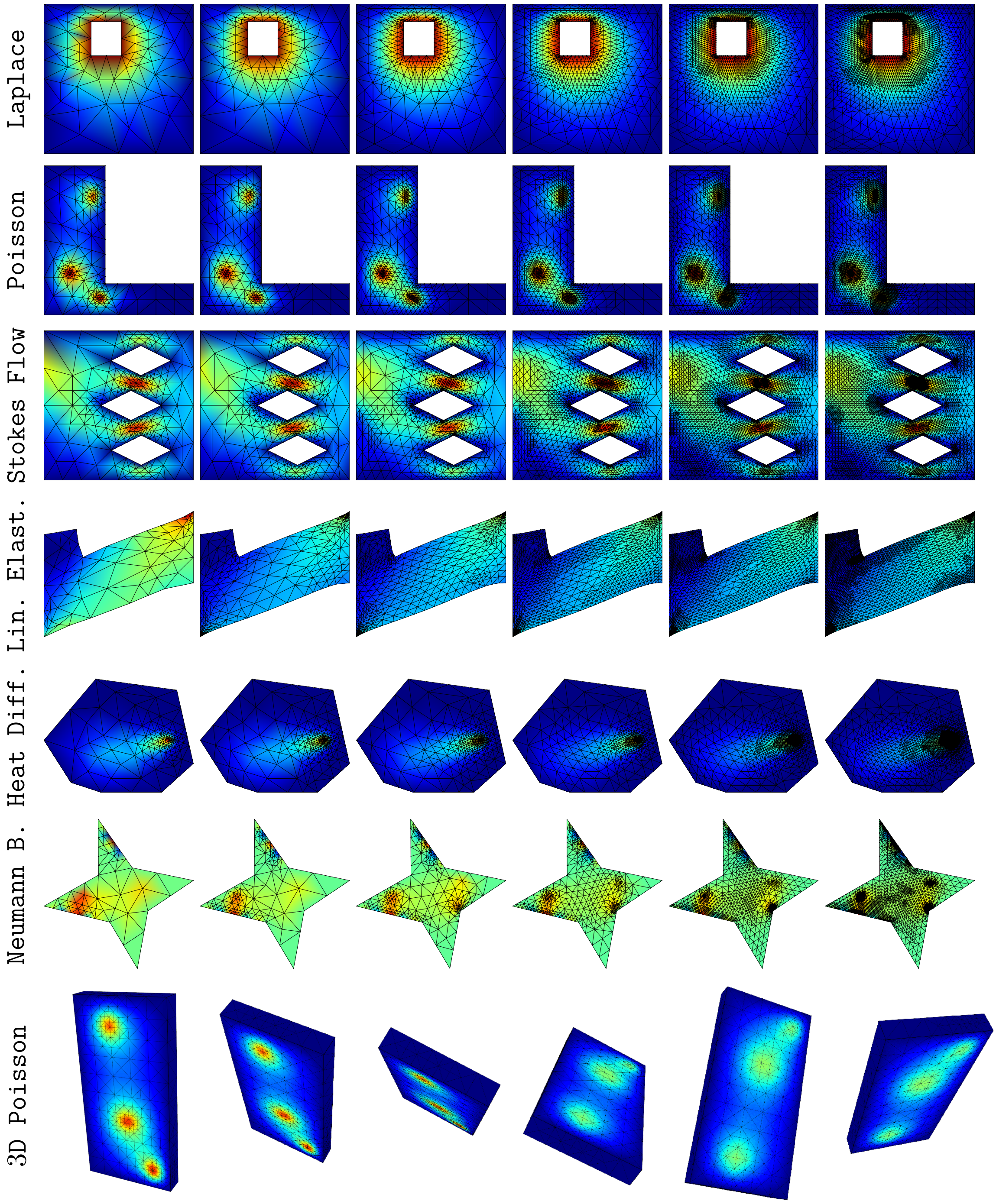


Figure B.9.: *ASMR++* meshes for the different environments and varying element penalties α. Each row uses the same policy conditioned on a range of low (**left**) to high (**right**) α values for the 2-dimensional environments. For `3D Poisson`, we use the same system of equations and α value, and show different camera angles.

C. Appendix for Chapter 5

C.1. Theoretical Convergence of the Iterative Mesh Generation Process

In this section, we provide a convergence proof of the iterative mesh generation process of *AMBER* in a simplified one-dimensional setting. We consider the unit interval as the domain of interest:

$$\Omega = [0, 1] \subset \mathbb{R}.$$

A one-dimensional mesh M is defined as a set of vertices

$$M = \{v_1, \dots, v_N\}$$

and elements (v_i, v_{i+1}). Let $v_1 = 0$, $v_N = 1$, and $v_i < v_{i+1}$ for all $i = 1, \dots, N-1$. The sizing field $f_e(M)$ induced by the mesh is directly related to the spacing between vertices and is defined for z such that $v_i \leq z < v_{i+1}$ as

$$f_e(M)(z) := v_{i+1} - v_i, \tag{C.1}$$

which is defined for the general setting in Section 5.3.2.

We construct a mesh generator g_{msh} that, given a sizing field $f : [0, 1] \to \mathbb{R}_{>0}$, generates a mesh as follows: set $v_1 := 0$, and define

$$v_{i+1} := \min(v_i + f(v_i), 1).$$

We terminate the process when $v_{i+1} = 1$, resulting in a mesh $g_{\text{msh}}(f) = \{v_1, \dots, v_N\}$. It is easy to see that

$$v_{i+1} = \sum_{j=1}^{i} f(v_j), \tag{C.2}$$

for intermediate points $i < N - 1$. Note that this generator acts as an inverse to the sizing field in the sense that

$$g_{\text{msh}}(f_e(M)) = M.$$

For $z = v_N = 1$, we set $f_e(M)(z) = v_N - v_{N-1}$.

Given a mesh $M^t = \{v_1^t, \dots, v_N^t\}$ and a target mesh M^*, we assume perfect predictions and define an interpolated sizing field as

$$\mathcal{I}_{M^t}(\hat{f})(z) := (1-d)\, f_e(M^*)(v_i^t) + d\, f_e(M^*)(v_{i+1}^t), \tag{C.3}$$

for $z = (1-d)\, v_i^t + d\, v_{i+1}^t$ with $0 \leq d \leq 1$.

Under these assumptions, we can prove the following:

Theorem C.1.1. *Let $M^1 = \{v_1^1, \dots v_{N_1}^1\}$ be an initial mesh and $M^* = \{v_1^*, \dots, v_N^*\}$ a target mesh. For a given mesh M^t, define one iteration of AMBER by*

$$M^{t+1} = g_{msh}(\mathcal{I}_{M^t}(\hat{f})). \tag{C.4}$$

Then, it holds that $M^N = M^*$.

Proof. We prove this by induction showing that the first k vertices of the k-th output $\{v_1^k, \dots v_k^k\} \subset M^k$ are equal to the target vertices $\{v_1^*, \dots v_k^*\} \subset M^*$.

The case for $k = 1$ is trivial. Consider now the $k+1$-th *AMBER* step. It holds for $i \leq k$:

$$\mathcal{I}_{M^k}(\hat{f})(v_i^k) = f_e(M^*)(v_i^k) = f_e(M^*)(v_i^*) = v_{i+1}^* - v_i^*, \tag{C.5}$$

using Equation C.3 for the first equality, the induction proposition for the second equality, and Equation C.1 for the last equality. From Equation C.2 and using the result from above, we get

$$v_{i+1}^{k+1} = \sum_{j=1}^{i} \mathcal{I}_{M^k}(\hat{f})(v_j^k) = \sum_{j=1}^{i} v_{j+1}^* - v_j^* = v_{i+1}^*, \tag{C.6}$$

for all $i \leq k$ which proves the desired result.

C.2. Datasets

We propose a total of six novel and varied datasets. `L-Shape` features L-shaped domains with a Gaussian Mixture Model as the load function and zero Dirichlet boundaries, adapted from the `Poisson` dataset proposed in *ASMR* (Freymuth et al., 2023) and *ASMR++* (Freymuth et al., 2024), as detailed in Section B.1.2. We vary the resolution of the expert mesh to define *easy*, *medium*, and *hard* variants. `Lattice` contains parameterized 2D lattices governed by the Laplace equation with complex Dirichlet boundary conditions, representative of structures used in, e.g., materials design. `Airfoil` includes flow simulations around airfoil-like shapes, as commonly encountered in aerodynamic engineering. `Beam` captures elasticity problems in mechanical engineering, using elongated beams with internal circular holes. The elongated beams induce long-range dependencies across the mesh. `Console` consists of 3D car seat crossmember geometries, parameterized and meshed by a human expert. The resulting meshes are optimized for downstream strength and durability

Table C.1.: Overview of dataset characteristics.

Name	Dim.	Application	Geometries	Online FEM	Expert Meshes	Process Conditions
L-Shape	2D	Electrostatics	Procedurally generated	Yes	Error indicator	Load function
Lattice	2D	Heat or fluid flow	Procedurally generated	Yes	Error indicator	Dirichlet boundary
Airfoil	2D	Fluid dynamics	Open-source dataset	No	Problem-specific heuristic	None
Beam	2D	Mechanical load	Procedurally generated	No	Problem-specific heuristic	None
Console	3D	Durability analysis	Closed-source dataset	No	Human labeled	None
Mold	3D	Injection molding	Open-source dataset	No	Human labeled	Inlet position

Table C.2.: Number of data points per split and min/mean/max number of vertices and elements per mesh in the training data. † For Mold, each geometry is paired with multiple inlet positions and corresponding expert meshes. Each of the 18 training geometries is used with 3 inlet positions. We reserve 5 Mold geometries for validation and test, using 1 and 2 inlet positions, respectively.

	# Data Points			# Vertices			# Elements		
Name	Train	Val	Test	Min	Mean	Max	Min	Mean	Max
L-Shape (*easy*)	20	20	20	387	549	674	705	1 042	1 292
L-Shape (*medium*)	20	20	20	1 562	2 234	2 736	2 985	4 358	5 365
L-Shape (*hard*)	20	20	20	9 951	13 224	15 884	19 563	26 185	31 510
Lattice	20	20	20	10 161	13 840	18 193	19 308	26 341	34 414
Airfoil	20	5	5	20 229	20 942	22 152	39 995	41 425	43 842
Beam	20	10	20	13 011	27 727	42 306	25 161	53 804	82 521
Console	19	2	5	2 222	6 606	10 130	7 800	25 769	41 856
Mold†	3×18	1×5	2×5	7 369	13 208	22 871	33 308	65 191	116 704

analyses. Mold represents injection molding setups with complex 3D plates, varying inlet positions, and handcrafted expert meshes.

Table C.1 summarizes dataset metadata. L-Shape and Lattice solve a concrete system of equations to yield a FEM solution over the mesh, as briefly explained in Section 2.3.1. This solution is used for expert mesh creation, and as features for the graph that we input into the MPN. For Mold, the process conditions $\mathcal{P}$ of each data point are comprised of the inlet position for the molding process, which always lies on the surface of the geometry. As such, we re-use each Mold geometry multiple times with different inlet positions, and generate a suitable expert mesh for each of them.

Table C.2 provides detailed statistics on mesh resolution for the training sets. Meshes range from 705 to 116 704 elements in the training data. The 3D datasets, i.e., Console and Mold, have a higher ratio of elements to vertices, as they use tetrahedral instead of triangular elements.

The sections below describe the construction of each dataset, including geometry generation and expert mesh creation. We implement the FEM for L-Shape and Lattice in scikit-fem (Gustafsson et al., 2020). For these datasets, we generate separate training data for each seed during training, but evaluate on a fixed set of validation and test data points. For the other datasets, we create a fixed set of training, validation and testing data points.

C.2.1. L-Shape

We consider adaptive, problem-specific meshes for Poisson's equation with zero Dirichlet boundary conditions, given in weak form as

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x = \int_\Omega qv \, \mathrm{d}x \quad \forall v \in V.$$

Each domain is a randomly generated L-shaped geometry of the form $\Omega = (0,1)^2 \setminus \left([p_0^{(1)}, 1] \times [p_0^{(2)}, 1]\right)$, with $p_0 = (p_0^{(1)}, p_0^{(2)})$ sampled from $\mathcal{U}(0.2, 0.8)^2$. The load function $q\colon \Omega \to \mathbb{R}$ is a GMM with three components. Means are drawn from $\mathcal{U}(0.0, 1.0)^2$ and resampled if they fall within 0.01 of the domain boundary or outside the domain. Covariances are initialized diagonally with log-uniform entries in $\exp(\mathcal{U}(\log 0.0001, \log 0.0005))$, and then randomly rotated to obtain full covariance matrices. Component weights follow $\exp(N(0,1)) + 1$, normalized across components, to provide meaningful weight to each component.

Expert meshes are constructed by refining a uniform coarse mesh with element volume 0.01 using a threshold-based heuristic that accounts for the load function and gradient jumps across element facets (Binev et al., 2004; Bangerth et al., 2012; Foucart et al., 2023). The local error indicator for element M_i is given by

$$\mathrm{err}(M_i) = h_i^2 \|q\|^2_{L^2(M_i)} + h_i \left\| [\![\nabla u \cdot \mathbf{n}]\!] \right\|^2_{L^2(\partial M_i)}, \tag{C.7}$$

where h_i denotes the characteristic length of M_i, and $[\![\nabla u \cdot \mathbf{n}]\!]$ denotes the jump in the normal derivative of u across facets of M_i, where $\mathbf{n}$ is the outward unit normal. This estimator highlights regions with strong source terms or large inter-element gradient discontinuities. Elements are marked for refinement if $\mathrm{err}(M_i) > \epsilon \cdot \max_j \mathrm{err}(M_j)$ with $\epsilon = 0.85$ fixed across all data points. Marked elements are refined via a conforming red-green-blue scheme (Carstensen, 2004), followed by Laplacian smoothing after each refinement step.

Each data point comprises a random domain, source term, and corresponding expert mesh. Additionally, we vary dataset difficulty by controlling the number of refinement steps. We use 25 steps for an *easy* variant, 50 steps for *medium*, and 100 for *hard*. We solve the equation on each intermediate mesh and extract the solution per vertex as a vertex-level input feature to our MPN. In addition, we use the evaluation of q at each vertex as a node feature. We use analogous features evaluated at pixel positions for the image baselines.

C.2.2. Lattice

The `Lattice` dataset emulates heat conduction or fluid transport through lattice structures during, e.g., compression-based manufacturing processes. It follows the same setup and refinement procedure as `L-Shape` (cf. Section C.2.1), but solves the Laplace equation

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x = 0 \quad \forall v \in V.$$

We impose a complex Dirichlet boundary condition based on a GMM, applied only to the inner boundary (i.e., the boundaries of the holes). The GMM has means sampled from $\mathcal{U}(0.1, 0.9)^2$ and covariances with diagonal entries drawn from $\exp(\mathcal{U}(\log 0.005, \log 0.01))$, followed by random rotation. The domain consists of a parameterized family of lattice-like geometries. Each instance contains a uniform grid of $k \times k$ square holes, with $k \in [5, 10]$ and hole sizes drawn from $\mathcal{U}(0.04, 0.075)$. Holes are placed evenly, ensuring uniform ligament thickness throughout the lattice.

The refinement procedure is identical to that used in `L-Shape`. Since there is no load function, i.e., $q{=}0$ for the Laplace equation, the local error indicator simplifies to

$$\text{err}(M_i) = h_i \left\| [\![\nabla u \cdot \mathbf{n}]\!] \right\|^2_{L^2(\partial M_i)} . \tag{C.8}$$

We use a fixed number of 100 refinement steps for all data points, corresponding to `L-Shape` (*hard*). Each data point consists of a domain, boundary condition, and expert mesh. We solve the equation on each intermediate mesh and use the solution at each vertex as an input feature to our MPN.

C.2.3. `Airfoil`

We sample airfoil geometries from the UIUC Airfoil Coordinates Database, each with a randomly selected angle of attack. Meshing is performed using GMSH-Airfoil-2D, which utilizes a problem-specific heuristic to generate high-quality meshes with large inflow/outflow regions and fine resolution near the airfoil. We generate 30 meshes, each placing the airfoil at the center of a circular domain within $[0, 1]^2$. The mesh size is set to 0.01 near the airfoil and 0.25 at the outer boundary, yielding approximately 20 000 vertices per mesh.

C.2.4. `Beam`

Beam geometries are widely used in mechanical engineering to study structural responses under load, for example in the context of non-linear elasticity (Lestringant et al., 2020; Antman, 2005). We generate adaptive beam geometries in GMSH (Geuzaine et al., 2009). We start with elongated rectangular domains, sampled from height h and length l, sampled as

$$h \sim \mathcal{N}(0.5, 0.05), \quad l \sim \mathcal{N}(10.0, 1.0).$$

Randomly placed disks are subtracted from the domain. The i-th disk has radius

$$r_i \sim \mathcal{U}(0.25h, 2.0h),$$

https://m-selig.ae.illinois.edu/ads/coord_database.html
https://github.com/cfsengineering/GMSH-Airfoil-2D/tree/main

and its center is placed at

$$x_i \sim \mathcal{U}(x_{i-1} + 1.5r_i, x_{i-1} + 20.0r_i), \quad y_i \sim \mathcal{U}(0, h),$$

using an initial reference position $x_0 = 0.1l$ to sample the first disk. Disk placement proceeds sequentially until the beam end is reached. A minimum part thickness of 0.001 is enforced. Meshing uses a manually crafted and carefully tuned heuristic that ensures fine resolution near disks and in thin regions of the geometry.

C.2.5. `Console`

`Console` uses data obtained from a real-world scenario in the automotive industry. We have a parameterized family of 3D geometries representing a car's seat crossmembers. The geometries are obtained using ONSHAPE and feature various sharp bends as well as up to two circular holes. Tetrahedral meshes for this dataset are generated by a human expert using ANSA. The expert is initially presented with a coarse mesh, on which they iteratively select regions to refine, specifying the target element size of each selected region. The resulting meshes are optimized for downstream strength and durability analyses, but our experiments are conducted solely on the meshes and their underlying geometry.

C.2.6. `Mold`

Injection molding is a key process for manufacturing thin, complex components in high-volume industrial settings (Rosato et al., 2012; Heaney, 2018). We select plane-like geometries from the ABC: A BIG CAD MODEL dataset (Koch et al., 2019), aligning them such that the longest dimension lies along x and the shortest along z. This standardization does not affect the rotation-invariant *AMBER*, but helps the *Image* baselines. Geometries are normalized so that the longest in-plane dimension is 1, and their thickness is rescaled to $z \sim \mathcal{U}(0.06, 0.09)$. Each geometry is duplicated three times with varying injection point locations, which are provided as process conditions and influence the meshing strategy. Geometries are imported into ABAQUS (Smith, 2009) and manually meshed by an expert using the standard tetrahedral meshing algorithm. Meshes are tailored for injection molding, with 4—6 elements across thickness and local refinement at holes, edges, and injection points. Mesh generation takes approximately 20 minutes per geometry, depending on complexity.

https://www.onshape.com/
https://www.beta-cae.com/ansa.htm

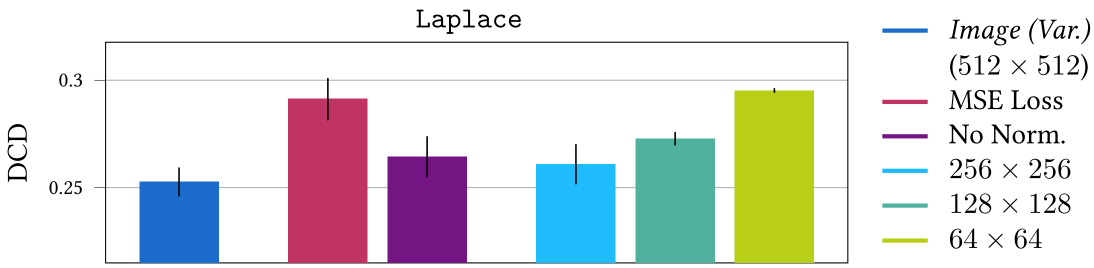


Figure C.1.: DCD comparison for different *Image (Variant)* parameter choices. Both the loss of Equation 2 and normalization are crucial for *Image (Variant)*. Performance improves with higher image resolutions, although the rate of improvement eventually slows down.

C.3. *AMBER* Hyperparameter

Table C.3 provides an overview of all relevant *AMBER* hyperparameters. We use the same hyperparameters across all datasets, except for the number of training steps, which we double for larger datasets to ensure convergence.

Table C.3.: *AMBER* parameters and experiment configuration.

Section	Parameter	Variable	Value
Optimization	Optimizer		ADAM
	Learning rate		1.0×10^{-3}
	Learning rate scheduler		linear with 10 % warm-up
	Weight decay		1.0×10^{-6}
MPN	Aggregation function	$\bigoplus$	mean
	MPN steps	L	20
	Activation function		LeakyReLU
	Edge dropout		0.1
	MLP layers		2
	Latent dimension		64
AMBER	Refinement steps	T	3
	Maximum buffer size		500 meshes
	Buffer addition frequency	k	8 samples every 128 batches
	Training steps		25 600 or 51 200
	Batch size		500 000 graph nodes plus edges
	Sizing field scaling	c_t	1.618^{T-t-1}

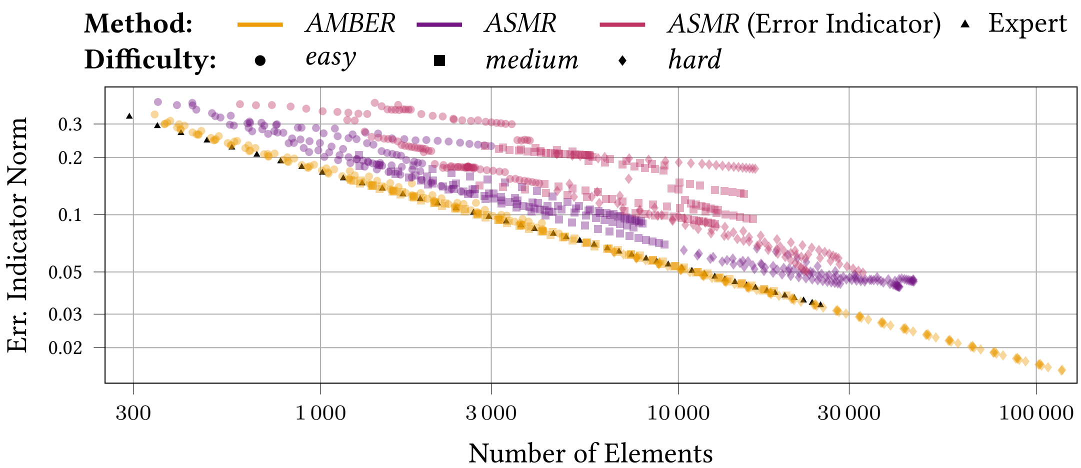


Figure C.2.: Log-log plot of error indicator norm versus number of mesh elements (lower left is better) for *AMBER*, *ASMR*, *ASMR (Error Indicator)* and the expert across `L-Shape` (*easy*, *medium*, *hard*). Each marker shows the mean over the test set for a given seed and target mesh resolution. This figure overlays Figure 5.5 with *ASMR (Error Indicator)*. We find that training *ASMR* on the indicator error yields less reliable, more noisy policies. However, as this *ASMR* variant is no longer constrained to a fixed refinement depth, it does not degrade as strongly for high-resolution meshes.

C.4. Extended Baseline Results

C.4.1. *Image* Experiments

We explore the behavior of the *Image (Variant)* baseline on `Lattice`. We vary image resolution and remove either input/output normalization (*Image (No Normalization)*) or the loss from Equation 5.2, replacing the latter with the MSE loss over the pixel-wise sizing fields (*Image (MSE Loss)*). Omitting both components recovers the original *Image* baseline. In all cases, we still use a softplus to transform the predictions, as we find that directly predicting a sizing field leads to worse performance and unstable mesh generation. Figure C.1 shows that performance improves with image resolution, although gains diminish at finer scales. Since the *Image (Variant)* enforces a uniform resolution by design, adapting to high-detail regions becomes prohibitively expensive, leading to substantial waste in less sensitive areas. In contrast, *AMBER* allows for variable sampling resolutions of the predicted sizing field by design, ensuring a more efficient prediction process, especially for highly adaptive meshes. Other than that, both normalization and our loss are crucial for accurate mesh generation, which is consistent with the *AMBER* parameter study in Section 5.6.1.

C.4.2. *ASMR (Error Indicator)*

We experiment with a version of *ASMR* that uses the error indicator in its reward function, i.e., sets $\text{err}(M_i^t)$ in Equation 4.7 to Equation C.7, leaving the rest of the reward unchanged. To further adapt the resulting *ASMR (Error Indicator)* version to our setup, we disable normalization of the initial errors, as we found this to be unstable when using the indicator, and adapt the MPN architecture to 10 message passing steps. We then increase the number of refinement steps to 7/9/11 for `L-Shape` (*easy*/*medium*/*hard*), allowing for elements with maximum refinement depth to be of the same size as the minimum expert elements.

Figure C.2 overlays Figure 5.5 with *ASMR (Error Indicator)*. We find that this method performs worse than *ASMR*, presumably because the error indicator is less expressive than the integrated reward. In comparison to the integrated reward, the indicator is noiser, yielding low relative contrast for elements of the same scale. This imbalance makes the reward function harder to optimize, reducing the consistency of the resulting policy. Yet, the indicator does not constraint the refinement depth, allowing for higher mesh resolutions and thus less saturation in simulation quality for finer meshes.

C.5. Visualizations

We provide additional visualizations for *AMBER* on all datasets, and for all methods on `L-Shape` (*hard*). We visualize the first test data point on the first trained seed for all methods. All visualizations include the expert mesh for reference, and zoom in on a representative region of the geometry.

C.5.1. Baseline Comparisons

Figure C.3 visualizes mesh outputs from all baseline methods and *AMBER* on the `L-Shape` (*hard*) dataset. *AMBER* produces more accurate and adaptive meshes, particularly in regions requiring fine detail and large variation in local mesh resolution. In contrast, the baseline methods exhibit artifacts or provide overly smooth or uniform sizing fields. For example, *ASMR* is constrained by the depth of its reference mesh, leading to too-uniform meshes, while *GraphMesh* (Khan et al., 2024) and the *Image* (Huang et al., 2021) baseline greatly over- and under-estimate local regions.

C.5.2. Full Rollouts

Figure C.4 and Figure C.5 illustrate full *AMBER* rollouts, showing the iterative mesh generation process from $t=0$ to $t=T=3$ across all datasets. Figure C.4 also visualizes the FEM solution on the expert for reference. Across datasets, each generation step incrementally refines the mesh, adding geometric detail and improving alignment with the target solution. The refinement constant c_t introduced in Section 5.3.3 ensures that early iterations produce coarser meshes, reducing computational cost in the initial stages.

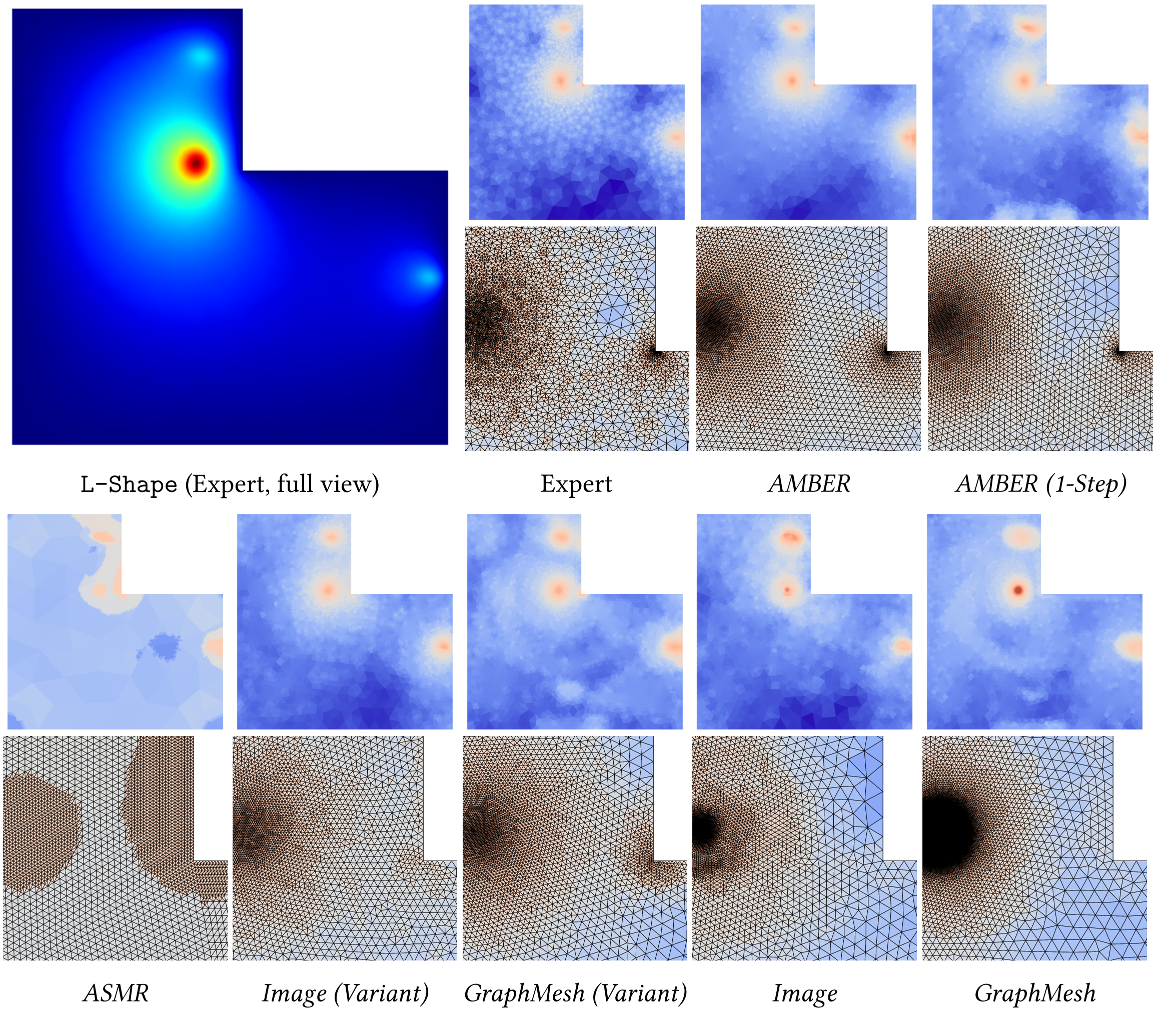


Figure C.3.: Expert mesh and generated meshes for all baseline methods and *AMBER* on the L-Shape (*hard*) dataset. The enlarged view of the expert mesh shows the FEM solution, while other plots show the element size, with red indicating smaller elements. *AMBER* yields more accurate and adaptive meshes, especially in regions with high resolution variability. *ASMR* has a constrained depth, leading to too-uniform refinements, and both *Image* (Huang et al., 2021) and *GraphMesh* (Khan et al., 2024) fail to correctly estimate sizing fields in local regions.

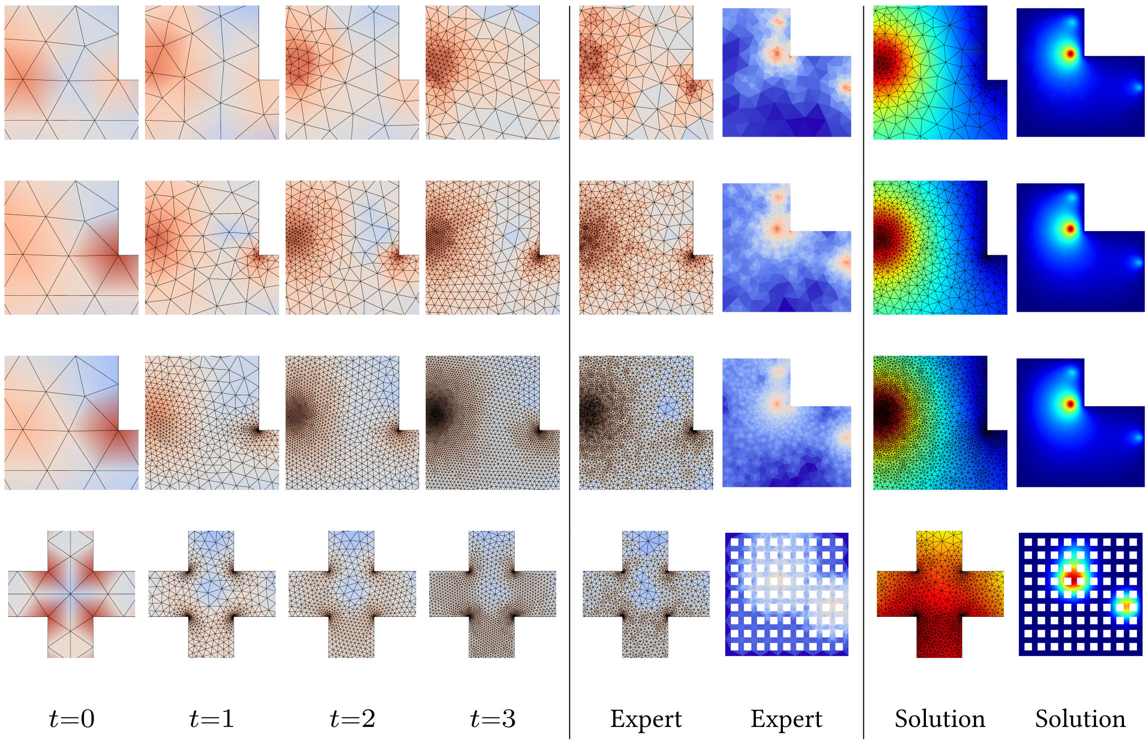


Figure C.4.: Close-ups of *AMBER* rollouts on the `L-Shape` (*easy*/*medium*/*hard*) and `Lattice` datasets from $t=0$ to $t=T=3$. Left, middle: For $t<3$, the colorscale denotes the prediction, otherwise the element size. Right: The colorscale shows the FEM solution on the zoomed-in and full domain for the expert. Successive *AMBER* steps produce increasingly refined meshes that better match the expert, improving sampling resolution for the next sizing field prediction. The refinement constant c_t from Section 5.3.3 controls mesh granularity over time, enabling coarse and efficient early steps.

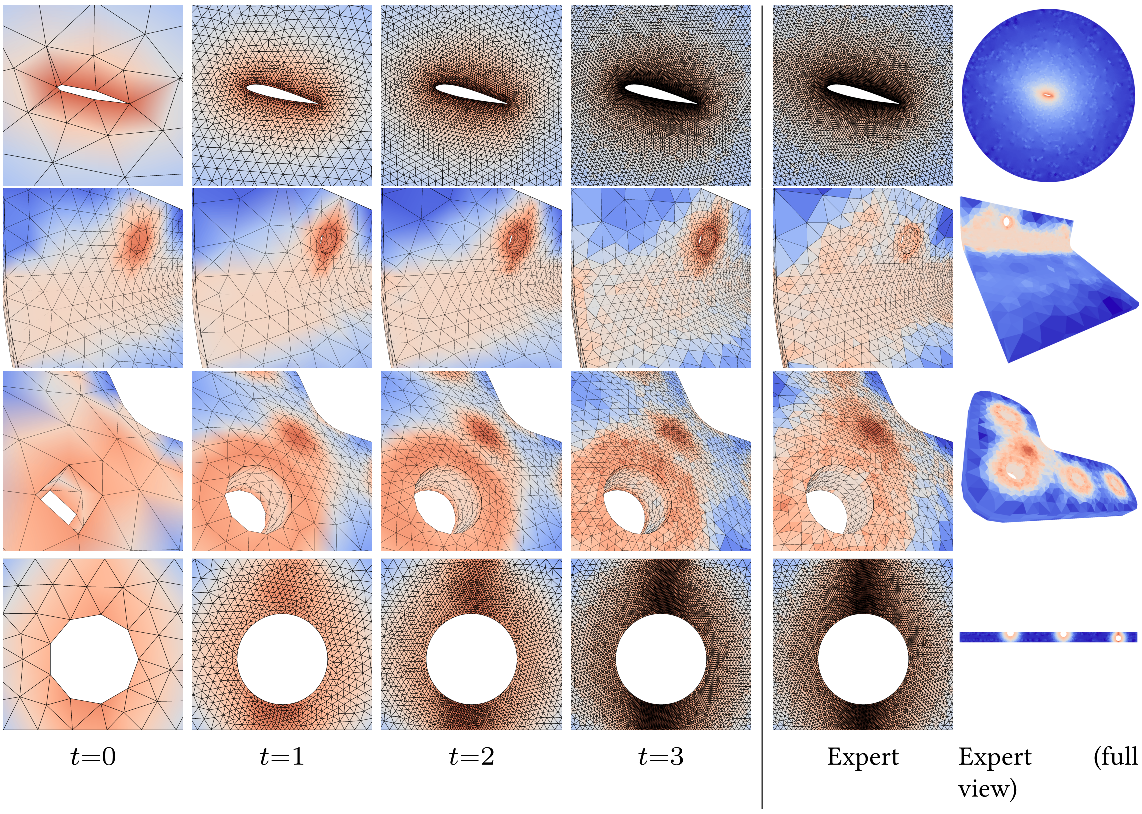


Figure C.5.: *AMBER* rollouts across the `Airfoil`, `Console`, `Mold` and `Beam`. The colorscale denotes predictions for $t<3$, and element size otherwise, with red indicating smaller values. As in Figure C.4, each step provides an increasingly detailed mesh, improving the next prediction's sampling resolution.

D. Appendix for Chapter 6

D.1. Derivations

For each output token o_t the trust region projection layer solves

$$\begin{aligned} &\underset{\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\arg\min} \quad \mathrm{KL}\left(\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) \parallel \tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})\right) \\ &\text{s.t.} \;\; \mathrm{KL}\left(\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) \parallel \pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})\right) < \epsilon \;\; \text{and} \;\; \sum_{o_t} \left[\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})\right] = 1. \end{aligned} \tag{D.1}$$

The first constraint enforces the trust region to the previous distribution $\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ and the second constraint ensures the resulting distribution is properly normalized. We solve the constrained optimization problem using the method of Lagrangian multipliers and start with the primal solution.

D.1.1. Primal Solution

To compute the primal solution of this optimization problem, we first set up the Lagrangian function by introducing Lagrangian multipliers $\eta > 0$ and λ for the first and second constraint, respectively. The corresponding Lagrangian is given as

$$\begin{aligned} &\mathcal{L}(\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}), \eta) \\ =&\mathrm{KL}\left(\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) \parallel \tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})\right) - \eta(\epsilon - \mathrm{KL}\left(\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) \parallel \pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})\right) \\ &- \lambda\left(1 - \sum_{o_t}\left[\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})\right]\right)) \\ =& -(\eta\epsilon + \lambda) + \sum_{o_t}\left[\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})\left(\log\frac{\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})} + \eta\log\frac{\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})} + \lambda\right)\right] \\ =& -(\eta\epsilon + \lambda) + \\ &\sum_{o_t}\left[\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})((\eta+1)\log\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) - \log\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) - \eta\log\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) + \lambda)\right]. \end{aligned} \tag{D.2}$$

We can now obtain the optimal primal solution to Equation D.1 by taking the derivative of the Lagrangian w.r.t. $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$, setting it to 0, and solving for $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$. The derivative is given by

$$\begin{aligned} &\frac{\partial\mathcal{L}(\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}), \eta)}{\partial\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})} \\ =&\sum_{o}\left[(\eta+1) + (\eta+1)\log\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) - (\log\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) + \eta\log\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})) + \lambda\right]. \end{aligned}$$

Clearly ${}^{\partial \mathcal{L}(\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}), \eta)}/_{\partial \pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})} = 0$ if all the individual terms of the sum are 0. Thus, the problem simplifies to

$$0 = (\eta + 1) + (\eta + 1) \log \pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) - (\log \tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) + \eta \log \pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})) + \lambda$$

which yields

$$\log \pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) = \frac{\log \tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) + \eta \log \pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) - (\eta + 1 + \lambda)}{\eta + 1} \tag{D.3}$$

and thus

$$\begin{aligned} \pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) &= \exp\left(\frac{\log \tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) + \eta \log \pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\eta + 1}\right) \exp\left(-\frac{\eta + 1 + \lambda}{\eta + 1}\right) \\ &\propto \exp\left(\frac{\log \tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) + \eta \log \pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\eta + 1}\right) \end{aligned} \tag{D.4}$$

Crucially, this primal solution allows computing a properly normalized distribution $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ without explicitly computing λ by replacing the exp in Equation D.4 with a softmax.

D.1.2. Dual Solution

The second step of the Lagrangian multiplier method is to solve the dual problem which finds the optimal dual parameters given the primal solution. To that end, we insert the primal solution from Equation D.3 into the Lagrangian (Equation D.2). Here most terms cancel out, leading to a dual of the form

$$D(\eta, \lambda) = -\eta\epsilon - \lambda - \eta - 1 = -\eta\epsilon - (\eta + 1 + \lambda). \tag{D.5}$$

In a second step toward a practically usable dual, we remove the dependency on λ by exploiting the constraint it enforces, i.e., $\sum_{o_t} [\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})] = 1$. Going to log space and again using Equation D.3, this property yields

$$\begin{aligned} 0 &= \log \sum_{o_t} [\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})] \\ &= \log \sum_{o_t} \left[\exp\left(\frac{\log \tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) + \eta \log \pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\eta + 1}\right) \exp\left(-\frac{\eta + 1 + \lambda}{\eta + 1}\right)\right] \\ &= -\frac{\eta + 1 + \lambda}{\eta + 1} + \log \sum_{o_t} \left[\exp\left(\frac{\log \tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) + \eta \log \pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\eta + 1}\right)\right] \end{aligned}$$

which we can rewrite as

$$\eta + 1 + \lambda = (\eta + 1) \log \sum_{o_t} \left[\exp\left(\frac{\log \tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) + \eta \log \pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\eta + 1}\right)\right]. \tag{D.6}$$

Now, inserting Equation D.6 into Equation D.5 removes the dependency on λ leading to

$$D(\eta) = -\eta\epsilon - (\eta + 1)\log \sum_{o_t} \left[\exp\left(\frac{\log \tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) + \eta \log \pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})}{\eta + 1} \right) \right].$$

Using this dual, we can find the optimal η^* by solving

$$\eta^* = \arg\max_\eta D(\eta) \;\; s.t. \;\; \eta \geq 0. \tag{D.7}$$

We can efficiently optimize this scalar optimization problem using the n-ary bracketing method described in Listing D.3.

D.1.3. Gradients

TROLL's trust region projections areis trivially differentiable using standard autograd tools, except for the numerical optimization of the dual to find the optimal step size η^*. Towards differentiating through the optimization in closed form, we change perspective and consider vectors q, $q^{(\log)}_{\text{old}}$, and $\tilde{q}^{(\log)}$ instead of the distributions $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$, $\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$, and $\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$. Here q corresponds to the probabilities of $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ while $q^{(\log)}_{\text{old}}$ and $\tilde{q}^{(\log)}$ denote to the normalized logits of $\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ and $\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$. We further assume all 3 vectors are normalized, i.e.,

$$\sum q = 1, \quad \sum \exp q^{(\log)}_{\text{old}} = 1 \quad \text{and} \quad \sum \exp \tilde{q}^{(\log)} = 1.$$

While this notation may seem slightly unintuitive at first, it simplifies the following derivations. As we assume the $\pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ and consequently $q^{(\log)}_{\text{old}}$ are constant, the only gradient we are interested in is $\partial\eta^*/\partial\tilde{q}^{(\log)}$, i.e., how the original LLM's output influences the optimal step size η^*. Since we do not have an analytical form for the optimal step size η^* but only the result of the numerical optimization, we need to introduce further analytical properties. Using the implicit differentiation (Dontchev et al., 2009) and differentiable matrix calculus (Magnus et al., 1989) techniques introduced to deep learning by OptNet (Amos et al., 2017), we start by writing out the Karush-Kuhn-Tucker (KKT) conditions (Karush, 1939) of the dual in Equation D.7 for the optimum at η^*. Denoting the Lagrangian multiplier corresponding to the $\eta \geq 0$ constraint by μ and realizing that

$$\nabla D(\eta) = \epsilon - \text{KL}\left(\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t}) \parallel \pi_{\text{old}}(o_t \mid \mathbf{q}, \mathbf{o}_{<t})\right) = \epsilon - q^T(\log q - q^{(\log)}_{\text{old}}),$$

the KKT are given by

$$\underbrace{\nabla D(\eta^*) + \mu\nabla(-\eta^*) = \epsilon - q^T(\log q - q^{(\log)}_{\text{old}}) - \mu = 0}_{\text{Stationarity}} \qquad \text{and} \qquad \underbrace{\mu(-\eta^*) = 0}_{\text{Complementary Slackness}}.$$

As there is no equality constraint in Equation D.7, primal feasibility is given by default. We can now take the total differentials around these conditions, which are given by

$$0 = d\left(\epsilon - q^T(\log q - q^{(\log)}_{\text{old}}) - \mu\right) = -d\left(q^T(\log q - q^{(\log)}_{\text{old}})\right) - d\mu = 0 \tag{D.8}$$

$$0 = d(\mu(-\eta^*) = d\mu(-\eta^*) + \mu(-d\eta^*), \tag{D.9}$$

where $d\epsilon$ vanishes as it is constant. Before proceeding, we need to rewrite the KL term $d\left(q^T(\log q - q^{(\log)}_{\text{old}})\right)$ in terms of $\tilde{q}^{(\log)}$ and simplify. First, we have

$$d\left(q^T(\log q - q^{(\log)}_{\text{old}})\right) = (1 + \log q - q^{(\log)}_{\text{old}})^T dq \tag{D.10}$$

and need to continue with the differential dq. Again using the primal solution Equation D.4, we get

$$q = \text{softmax}\left(\frac{\eta^* q^{(\log)}_{\text{old}} + \tilde{q}^{(\log)}}{\eta^* + 1}\right). \tag{D.11}$$

Assuming the old logits are a constant, we can write the corresponding differential as

$$dq = \frac{\partial q}{\partial \tilde{q}^{(\log)}} d\tilde{q}^{(\log)} + \frac{\partial q}{\partial \eta^*} d\eta^*.$$

Inserting this term into Equation D.10 and the plugging the result into Equation D.8 and Equation D.9 yields

$$-\left(1 + \log q - q^{(\log)}_{\text{old}}\right)^T \frac{\partial q}{\partial \eta^*} d\eta^* - d\mu = \left(1 + \log q - q^{(\log)}_{\text{old}}\right) \frac{\partial q}{\partial \tilde{q}^{(\log)}} d\tilde{q}^{(\log)} \tag{D.12}$$

$$-\mu d\eta^* - \eta^* d\mu = 0, \tag{D.13}$$

which we can use to compute the desired gradient $\frac{\partial \tilde{q}^{(\log)}}{\partial \eta^*}$. We consider two separate cases. First, if the original KL trust region is not violated, then $\eta^* = 0$ and $\mu > 0$. In this case, Equation D.13 directly yields that $d\eta^* = 0$ and thus the entire gradient $\frac{\partial \eta^*}{\partial \tilde{q}^{(\log)}}$ is zero. Second, the original KL trust region constraint is active and thus $\eta^* > 0$ and $\mu = 0$. In this case Equation D.13 gives $d\mu = 0$ which simplifies Equation D.12. Reordering the remaining terms gives the required gradient

$$\frac{\partial \eta^*}{\partial \tilde{q}^{(\log)}} = \frac{1}{-(1 + \log q - q^{(\log)}_{\text{old}})^T \frac{\partial q}{\partial \eta^*}} (1 + \log q - q^{(\log)}_{\text{old}})^T \frac{\partial q}{\partial \tilde{q}^{(\log)}}$$

The required partial derivatives can be obtained from Equation D.11 as

$$\frac{\partial q}{\partial \tilde{q}^{(\log)}} = \frac{1}{\eta^* + 1}(\text{D}(q) - qq^T) \quad \text{and} \quad \frac{\partial q}{\partial \eta^*} = \frac{1}{(\eta + 1)^2}(\text{D}(q) - qq^T)(q^{(\log)}_{\text{old}} - \tilde{q}^{(\log)}),$$

where $D(q)$ denotes a diagonal matrix with the entries of q on the diagonal.

Crucially, for practical purposes, we never need to explicitly materialize the matrices in the partial derivatives. The resulting backward introduces negligible computational and memory overhead. Listing D.2 shows that, in the non-sparsified case, the backward can be written in less than ten lines of python code.

D.1.4. Sparsification

Theorem D.1.1. *For any pair of logits* $o_t^{(1)}$ *and* $o_t^{(2)}$, *with* $\tilde{\pi}_\theta(o_t^{(1)} \mid \mathbf{q}, \mathbf{o}_{<t}) \geq \tilde{\pi}_\theta(o_t^{(2)} \mid \mathbf{q}, \mathbf{o}_{<t})$ *w.l.o.g., the logit-wise terms that sum to the KL are equally ordered*

$$\tilde{\pi}_\theta(o_t^{(1)} \mid \mathbf{q}, \mathbf{o}_{<t}) \log \frac{\tilde{\pi}_\theta(o_t^{(1)} \mid \mathbf{q}, \mathbf{o}_{<t})}{\pi_{old}(o_t^{(1)} \mid \mathbf{q}, \mathbf{o}_{<t})} \quad \geq \quad \tilde{\pi}_\theta(o_t^{(2)} \mid \mathbf{q}, \mathbf{o}_{<t}) \log \frac{\tilde{\pi}_\theta(o_t^{(2)} \mid \mathbf{q}, \mathbf{o}_{<t})}{\pi_{old}(o_t^{(2)} \mid \mathbf{q}, \mathbf{o}_{<t})} \tag{D.14}$$

iff $e^\kappa \geq \gamma$, *where* $\kappa = \frac{\tilde{\pi}_\theta(o_t^{(1)} \mid \mathbf{q}, \mathbf{o}_{<t})}{\tilde{\pi}_\theta(o_t^{(2)} \mid \mathbf{q}, \mathbf{o}_{<t})}$ *is the current probability ratio of the pair and* γ *in* $\frac{\pi_{old}(o_t^{(1)} \mid \mathbf{q}, \mathbf{o}_{<t})}{\pi_{old}(o_t^{(2)} \mid \mathbf{q}, \mathbf{o}_{<t})} = \gamma\kappa$ *gives the multiplier of the old ratio.*

Proof. Rewrite $\tilde{\pi}_\theta(o_t^{(1)} \mid \mathbf{q}, \mathbf{o}_{<t}) \geq \tilde{\pi}_\theta(o_t^{(2)} \mid \mathbf{q}, \mathbf{o}_{<t})$ as $p(x_1) = \kappa \cdot p(x_2)$ for $\kappa \geq 1$ using $p(x_i) = \tilde{\pi}_\theta(o_t^{(i)} \mid \mathbf{q}, \mathbf{o}_{<t})$ for clarity and similarly replace $q(x_i) = \pi_{\text{old}}(o_t^{(i)} \mid \mathbf{q}, \mathbf{o}_{<t})$. Then compare the contributions of x_1 and x_2 to the KL divergence

$$\kappa p(x_2) \log \frac{\kappa p(x_2)}{q(x_1)} \geq p(x_2) \log \frac{p(x_2)}{q(x_2)}$$
$$\kappa \log k \geq \log \frac{q(x_1)}{q(x_2)}$$

and substitute $\frac{q(x_1)}{q(x_2)} =: \gamma \frac{p(x_1)}{p(x_2)} = \gamma\kappa$

$$e^\kappa \kappa \geq \gamma \frac{p(x_1)}{p(x_2)}$$
$$e^\kappa \geq \gamma.$$

Here, the assumption that the relative likelihood κ of $o_t^{(1)}$ and $o_t^{(2)}$ was not exponentially larger before usually holds in practice, as the token distributions are pushed farther from uniform during training (Cui et al., 2025b).

Definition D.1.2. For any subset $\mathcal{S}$ of the possible tokens, we define $p_\mathcal{S}$, or just p' when the mask is clear, as the *sparsed* distribution. For tokens not in $\mathcal{S}$, it has default probability p_d and the same probability as p for all others up to the renormalization constant.

$$p_\mathcal{S}(x) = p'(x) := \begin{cases} \gamma p(x), & \text{for } x \in \mathcal{S} \\ p_d, & \text{else} \end{cases}, \qquad \gamma = \frac{1 - (|\mathcal{V}| - |\mathcal{S}|) \cdot p_d}{\sum_{x \in \mathcal{S}} p(x)}. \tag{D.15}$$

The renormalization factor γ accounts for the previous total mass $\sum_{x \in \mathcal{S}} p(x)$ of the kept tokens and new mass $(|\mathcal{V}| - |\mathcal{S}|) \cdot p_d$ of the dropped tokens.

In the case of equal sparsification masks for distributions p, q, we can prove a practically tight upper bound for the true divergence $\mathrm{KL}(p \parallel q)$ in terms of the sparse divergence $\mathrm{KL}(p' \parallel q')$.

Theorem D.1.3. *Let p, q be categorical distributions over the vocabulary $|\mathcal{V}|$ with identical top-k logits,* $\mathrm{topk(p)} = \mathrm{topk(q)}$ *and* $\sum_{x \in \mathrm{topk(p)}} p(x) = \sum_{x \in \mathrm{topk(q)}} q(x) = 1 - \delta$*, i.e., equal total probability. Then the sparsed distributions p', q' with density*

$$p'(x) := \begin{cases} \gamma p(x), & \text{for } x \in \mathrm{topk(p)} \\ p_d, & \text{else} \end{cases}, \quad q'(x_i) := \begin{cases} \gamma q(x), & \text{for } x \in \mathrm{topk(q)} \\ p_d, & \text{else} \end{cases},$$

and normalization constant $\gamma(\delta, k, |\mathcal{V}|, p_d) \approx 1$ follow the inequality

$$KL(p \parallel q) \leq \gamma^{-1} KL(p' \parallel q') + \delta \log \frac{\delta}{q_{\min}},$$

where $q_{\min} = \arg\min_x q(x)$.

Proof. Rename the logits in descending order of probability under p, such that $p(x_0) \geq p(x_1) \geq \cdots \geq p(x_{|\mathcal{V}|-1})$. Assume there is $k < |\mathcal{V}|$ such that the largest k logits of both p and q have exactly the total probability mass $\sum_{i=0}^{k} p(x_i) = \sum_{i=0}^{k-1} q(x_i) = 1 - \delta$ and the subset of largest logits is identical. Every nondegenerate distribution has $q_{\min} \leq q(x_i)$ and all $p(x_i) \leq \delta$ for $i \geq k$, as the total mass could otherwise not be $1 - \delta$. So split the sum over logits in the KL divergence and apply both inequalities

$$\begin{aligned} \mathrm{KL}(p \parallel q) &= \sum_{i=0}^{k-1} p(x_i) \log \frac{p(x_i)}{q(x_i)} + \sum_{i=k}^{|\mathcal{V}|-1} p(x_i) \log \frac{p(x_i)}{q(x_i)} \\ &\leq \sum_{i=0}^{k-1} p(x_i) \log \frac{p(x_i)}{q(x_i)} + \sum_{i=k}^{|\mathcal{V}|-1} p(x_i) \log \frac{\delta}{q(x_i)} \\ &\leq \sum_{i=0}^{k-1} p(x_i) \log \frac{p(x_i)}{q(x_i)} + \sum_{i=k}^{|\mathcal{V}|-1} p(x_i) \log \frac{\delta}{q_{\min}} \\ &= \sum_{i=0}^{k-1} p(x_i) \log \frac{p(x_i)}{q(x_i)} + \log \frac{\delta}{q_{\min}} \underbrace{\sum_{i=k}^{|\mathcal{V}|-1} p(x_i)}_{=\delta} \\ &= \sum_{i=0}^{k-1} p(x_i) \log \frac{p(x_i)}{q(x_i)} + \delta \log \frac{\delta}{q_{\min}}. \end{aligned}$$

Replace p, and analogously q, with their sparsed version as defined in Theorem D.1.2,

$$p'(x_i) := \begin{cases} \gamma p(x_i), & \text{for } i < k \\ p_d, & \text{for } i \geq k \end{cases}, \tag{D.16}$$

where $\gamma = \frac{1-(|\mathcal{V}|-k)\cdot p_d}{(1-\delta)}$ renormalizes the $(1-\delta)$ mass of the selected tokens to account for the default mass $(|\mathcal{V}|-k)\cdot p_d$ of the sparsified tokens. Multiplying with ones and adding a zero to the KL bound yields the relation to the sparse KL,

$$\begin{aligned}
\mathrm{KL}\left(p \parallel q\right) &\leq \sum_{i=0}^{k-1} \frac{\gamma}{\gamma} p(x_i) \log \frac{\gamma p(x_i)}{\gamma q(x_i)} + \delta \log \frac{\delta}{q_{\min}} + \gamma^{-1} \sum_{i=k}^{|\mathcal{V}|-1} p'(x_i) \underbrace{\log \frac{p_d}{p_d}}_{=0} \\
&= \gamma^{-1} \sum_{i=0}^{k-1} p'(x_i) \log \frac{p'(x_i)}{q'(x_i)} + \delta \log \frac{\delta}{q_{\min}} + \gamma^{-1} \sum_{i=k}^{|\mathcal{V}|-1} p'(x_i) \log \frac{p'(x_i)}{q'(x_i)} \\
&= \gamma^{-1} \mathrm{KL}\left(p' \parallel q'\right) + \delta \log \frac{\delta}{q_{\min}}.
\end{aligned}$$

Assuming that q's probabilities can be represented by normal single precision IEEE-754 numbers, $q_{\min} > 1.17549 \cdot 10^{-38}$, and $k \ll |\mathcal{V}|$, e.g. $k = 256$ of vocab size $|\mathcal{V}| = 151936$ while using threshold $\delta = 10^{-5}$ and default mass $p_d = 10^{-12}$, the sparse KL approximation,

$$\begin{aligned}
\mathrm{KL}\left(p \parallel q\right) &\leq \frac{(1-\delta)}{1-(|\mathcal{V}|-k)\cdot p_d} \mathrm{KL}\left(p' \parallel q'\right) + \delta \log \frac{\delta}{q_{\min}} \\
&= \frac{0.99999}{1 - 151680 \cdot 10^{-12}} \mathrm{KL}\left(p' \parallel q'\right) + 10^{-5} \log \frac{10^{-5}}{1.17549 \cdot 10^{-38}} \\
&\leq 0.99999015168 \cdot \mathrm{KL}\left(p' \parallel q'\right) + 0.00075823623,
\end{aligned}$$

is accurate enough for limiting the true divergence to values on the order of 0.05 as

$$\begin{aligned}
\mathrm{KL}\left(p \parallel q\right) &\leq 0.99999015168 \cdot \mathrm{KL}\left(p' \parallel q'\right) + 0.00075823623 \\
&\leq 0.99999015168 \cdot 0.05 + 0.00075823623 \\
\mathrm{KL}\left(p \parallel q\right) &\leq 0.050757743814.
\end{aligned}$$

D.2. Implementation

While the theoretical derivation of the differentiable trust region projection can look complex, the final implementation is fairly straightforward. We give `PyTorch`-adjacent pseudocode for the dense variant of the primal (Listing D.1) and dual (Listing D.2) in the following. Note that the sparse implementation mostly differs in the usage of a custom sparse tensor class that maintains a default probability for the implicit entries. While this requires additional care in terms of indexing and allows for optimizations of, e.g., the KL computation, the general logic remains unchanged. Listing D.3, shows our n-ary bracketing method to optimize the dual.

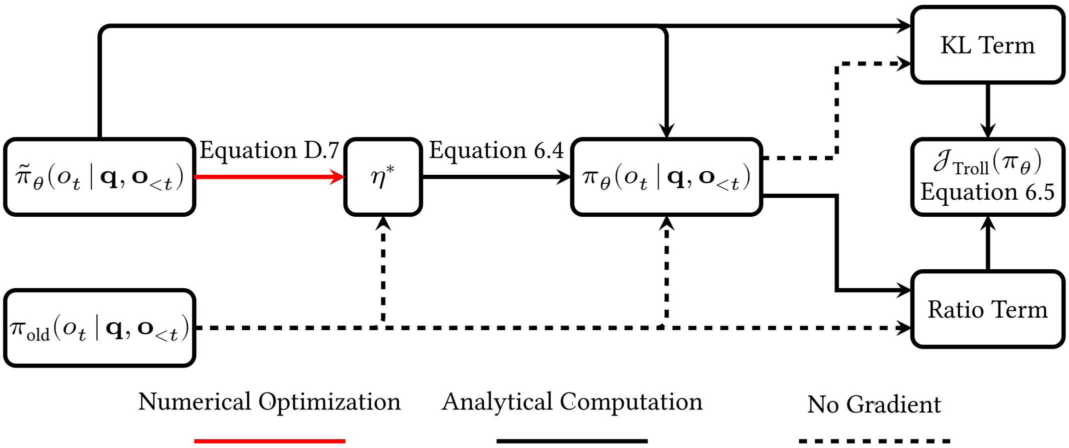


Figure D.1.: Compute graph from the LLM output $\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ to the RL objective $\mathcal{J}_{\text{Troll}}(\pi_\theta)$. First, the optimal step size η^* is computed for each token. Here, we use the fact that the step size for tokens that do not violate the trust region is trivially 0. For the tokens that do violate the trust region, we need to optimize a 1D convex optimization problem to compute η^*. Next, the projection computes the optimal distribution within the trust region, $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$, which is then used in the objective. This objective combines the standard policy ratio term from PPO with a KL term to regress $\tilde{\pi}_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$ toward $\pi_\theta(o_t \mid \mathbf{q}, \mathbf{o}_{<t})$.

```python
def TROLLProjection(log_target_prob, log_ref_prob, bound):
    kl_div = (log_target_prob.exp() * (log_target_prob -
        log_ref_prob)).sum(dim=-1)
    needs_projection = kl_div >= bound # only projects where
        necessary
    # ... masking of needed tokens
    # solve dual problem, i.e. find $\eta^*$
    opt_eta = DualSolver(log_target_prob, log_ref_prob, bound)
    primal_unnormalized = (opt_eta * log_ref_prob +
        log_target_prob) / (opt_eta + 1)
    primal = inner.log_softmax(dim=-1)
    # ... combine masked unprojected and primal logits into one
    return projected_logits
```

Listing D.1: *TROLL*'s differentiable projection only calls a differentiable dual solver and otherwise uses standard autodiff operations.

D.3. Experimental Setup

D.3.1. Models

Table D.1 lists all model checkpoints used in this work. They are publicly available and can be downloaded under the provided links. We used the thinking mode for the models from the Qwen3-Family. For the non-instruct versions of Llama-3.1, Llama-3.2, and Apertus, we used the chat templates from the respective instruct versions.

```python
def DualSolver.forward(log_target_prob, log_ref_prob, bound):
    # define objective in terms of log eta (such that eta > 0)
    opt_log_eta = optimize1d(
            lambda log_eta: dual(log_eta, ...),
            # ... bounds and termination config
        )
    # ... save for backward
    return opt_log_eta.exp()

def dual(log_eta, bound, log_target_prob, log_ref_prob)
    eta = log_eta.exp()
    inner = (log_target_prob + eta * log_ref_prob) / (eta + 1)
    inner_lse = logsumexp(inner, axis=-1)
    # negative of objective, since we minimize
    return eta *  bound + (eta + 1) * inner_lse

def DualSolver.backward(grad_output):
    # ... recompute primal = ... as in TROLLProjection
    one_plus_logratio = 1 + primal.log() - log_ref_prob
    # compute one_plus_logratio.T @ dprimal_dlog_output implicitly
    numerator = primal * (one_plus_logratio - vecdot(primal, one_plus_logratio).unsqueeze(-1) / (opt_eta + 1)
    # compute  dprimal_dopt_eta implicitly
    diff = log_ref_prob - log_target_prob
    dprimal_dopt_eta = primal * (diff - vecdot(primal, diff).unsqueeze(-1) / (opt_eta + 1)**2
    return grad_output * (numerator / -vecdot(one_plus_logratio, dprimal_dopt_eta))
```

Listing D.2: Custom forward and backward code for *TROLL*'s dual solver.

D.3.2. Datasets

Table D.2 provides links for all datasets their and software repositories used in this work, along with their download links. Listing D.4 shows the system prompt used for the `DAPO`-Train, `DAPO`-Eval, and `Math`-Eval experiments.

DAPO-Math

D.3.3. Training Hyperparameters

We provide hyperparameters for our training setup in Table D.3. We maintain consistent hyperparameters across all experiments, except for Section 6.6.1, where we always vary exactly one parameter.

```
class Optimizer1D:

    def batched_linspace(lower, upper, num_points):
        # Batched linspace: lower and upper are (batch_size, 1), returns (batch_size, num_points)

        steps = linspace(0, 1, num_points)
        return lower + (upper - lower) * steps

    def _opt_step(func, x, lower, upper):
        batch_size, num_points = x.shape
        # batched evaluation of all points
        y = func(x)
        # select min index for each batch element
        min_idx = argmin(y, dim=1)

        # take left and right point
        l_idx = min_idx - 1
        u_idx = min_idx + 1
        l_tmp = x[arange(batch_size), clamp(l_idx, 0, num_points - 1)]
        u_tmp = x[arange(batch_size), clamp(u_idx, 0, num_points - 1)]
        new_lower = where(l_idx < 0, lower), l_tmp)
        new_upper = where(u_idx >= num_points, upper, u_tmp)
        return new_lower, new_upper

    def optimize(func, lower, upper, num_points, max_steps, x_threshold):
        # batched, parallel, gradient-free, optimization of a 1D function

        l, u = lower, upper
        # refine lower and upper until convergence
        for step in range(max_steps):
            x = Optimizer1D.batched_linspace(l, u, num_points + 2)
            x = x[:, 1:-1]

            l, u = Optimizer1D._opt_step(func, x, l, u)

            if ((l - u) < x_threshold).abs().all():
                break

        x = (l + u) / 2

        return x
```

Listing D.3: Exemplary pseudocode for N-ary bracketing search.

Model	Link
Qwen3-0.6B	`https://huggingface.co/Qwen/Qwen3-0.6B`
Qwen3-1.7B	`https://huggingface.co/Qwen/Qwen3-1.7B`
Qwen3-4B	`https://huggingface.co/Qwen/Qwen3-4B`
Qwen3-8B	`https://huggingface.co/Qwen/Qwen3-8B`
Qwen3-14B	`https://huggingface.co/Qwen/Qwen3-14B`
Qwen2.5-0.5B-Instruct	`https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct`
Qwen2.5-1.5B-Instruct	`https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct`
Qwen2.5-3B-Instruct	`https://huggingface.co/Qwen/Qwen2.5-3B-Instruct`
Qwen2.5-7B-Instruct	`https://huggingface.co/Qwen/Qwen2.5-7B-Instruct`
Llama-3.1.8B	`https://huggingface.co/meta-llama/Llama-3.1-8B`
Llama-3.1.8B-Instruct	`https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct`
Llama-3.2-3B	`https://huggingface.co/meta-llama/Llama-3.2-3B`
LLama-3.2-3B-Instruct	`https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct`
FineMath-Llama 3B	`https://huggingface.co/HuggingFaceTB/FineMath-Llama-3B`
Apertus-8B	`https://huggingface.co/swiss-ai/Apertus-8B-2509`
Apertus-8B-Instruct	`https://huggingface.co/swiss-ai/Apertus-8B-Instruct-2509`
SmolLM3-3B	`https://huggingface.co/HuggingFaceTB/SmolLM3-3B`

Table D.1.: Pre-trained model checkpoints used as starting points for finetuning throughout this work.

```
Your task is to follow a systematic, thorough reasoning process
before providing the final solution. This involves analyzing,
summarizing, exploring, reassessing, and refining your thought
process through multiple iterations. Structure your response into
 two sections: Thought and Solution. In the Thought section,
present your reasoning using the format: "<think> {thoughts} </
think>".
```

Listing D.4: Common system prompt for `DAPO`-Train, `DAPO`-Eval, and `Math`-Eval

D.4. Additional Results

D.4.1. Detailed Qwen Results on `Math`-Eval

Table D.4 provides success rates for individual `Math` evaluation datasets for Qwen2.5-7B-Instruct and Qwen3-8B models trained on `DAPO` with different advantage estimation methods.

Table D.2.: Summary of dataset and reward manager links

Category	Item	URL
DAPO-Math	Dataset	`https://github.com/PRIME-RL/Entropy-Mechanism-of-RL`
GSM8k	Dataset	`https://huggingface.co/datasets/openai/gsm8k`
Eurus (Math)	Dataset	`https://huggingface.co/datasets/PRIME-RL/Eurus-2-RL-Data`
Eurus (Code)	Dataset	`https://huggingface.co/datasets/PRIME-RL/Eurus-2-RL-Data`
Eurus (Code)	Codeforces Dataset	`https://huggingface.co/datasets/MatrixStudio/Codeforces-Python-Submissions`
Eurus (Code)	Reward Manager	`https://github.com/bytedance/SandboxFusion`

Hyperparameter	Variable	Value
Trust Region Size	ϵ	0.05
KL Regression Factor	α	1.0
Sparsity Remaining Mass	$1-\delta$	0.99999
Max. Sparse Tokens	K	64
Chunk Size		1024
Clip Value	ϵ_{ppo}	0.2
Learning Rate		10^{-6}
Gradient Max Norm		1.0
Weight Decay		0.0
Learning Rate-Schedule		constant
Learning Rate Critic (PPO only)		10^{-5}
Weight Decay Critic (PPO only)		0.01
Sampler Per Query		8
Batch Size		32
Batches Per Step		8

Table D.3.: Hyperparameters. We use these parameters for all experiments unless mentioned otherwise.

D.4.2. Additional Models

Figure D.2 and Figure D.3 show success rates for different 3B and 8B models, respectively. We find that *TROLL* causes some models, such as Finemath-3B, Llama3.2-3B and Llama3.1-8B to receive a training signal in significantly fewer steps. Other models, such as Apertus-8B show more stable performance when trained with *TROLL*. Finally, for models that work well with the *Clip* objective, using *TROLL* generally yields some performance benefit even though the success rates on `GSM8K` are almost saturated.

D.4.3. Qwen3 on `Eurus-Math` and `GSM8K`

We additionally evaluate different Qwen3 model sizes on `GSM8K` in Figure D.4, finding that most models quickly saturate on this comparatively easy task. Nevertheless, using *TROLL* instead of *Clip* generally provides a small boost in performance across model sizes.

Method		AIME24	AIME25	AMC	MATH	Omni-Math	Olympiad	Minerva
				Qwen2.5-7B-Instruct				
GRPO	*Clip*	0.066	0.075	0.535	0.683	0.239	0.286	0.304
	TROLL	0.168	0.129	0.587	0.712	0.254	0.317	0.284
Dr.GRPO	*Clip*	0.103	0.067	0.560	0.662	0.242	0.288	0.295
	TROLL	0.168	0.135	0.605	0.706	0.259	0.320	0.317
PPO	*Clip*	0.092	0.064	0.503	0.706	0.251	0.316	0.299
	TROLL	0.162	0.093	0.547	0.734	0.258	0.320	0.332
GSPO	*Clip*	0.026	0.002	0.188	0.344	0.106	0.102	0.120
	TROLL	0.159	0.076	0.531	0.699	0.257	0.297	0.310
RF++	*Clip*	0.096	0.033	0.556	0.666	0.239	0.291	0.299
	TROLL	0.166	0.102	0.552	0.687	0.263	0.320	0.319
				Qwen3-8B				
GRPO	*Clip*	0.439	0.293	0.720	0.889	0.465	0.547	0.431
	TROLL	0.547	0.353	0.790	0.812	0.465	0.497	0.391
Dr.GRPO	*Clip*	0.458	0.305	0.743	0.891	0.477	0.547	0.425
	TROLL	0.447	0.337	0.769	0.880	0.466	0.522	0.403
PPO	*Clip*	0.380	0.234	0.694	0.874	0.439	0.531	0.405
	TROLL	0.524	0.408	0.780	0.910	0.521	0.567	0.425
GSPO	*Clip*	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	TROLL	0.474	0.407	0.813	0.897	0.514	0.547	0.408
RF++	*Clip*	0.356	0.251	0.728	0.897	0.459	0.538	0.415
	TROLL	0.490	0.355	0.803	0.891	0.528	0.563	0.415

Table D.4.: Success rates for individual `Math` test datasets for Qwen2.5-7B-Instruct and Qwen3-8B models trained on `DAPO` with different advantage estimation methods. *TROLL* provides consistent benefits across methods and evaluation tasks, showing well-balanced improvements in performance. It also successfully trains GSPO without divergence, wheres *Clip* eventually causes unstable updates, as shown in Figure 6.6 and Figure 6.7.

Similarly, Figure D.5 shows that *TROLL* leads to improvements for Qwen3-8B trained with GRPO on `Eurus-Math`.

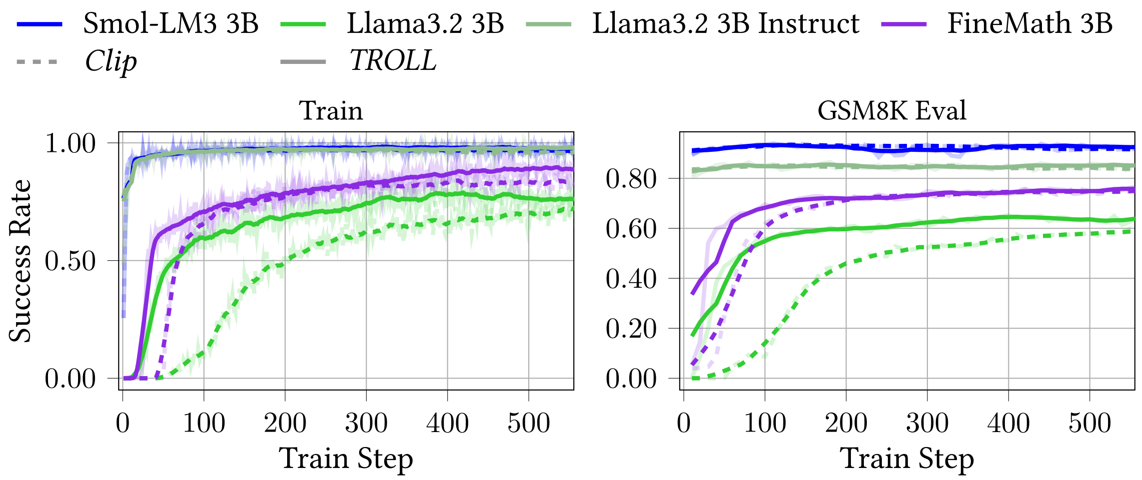


Figure D.2.: *TROLL* and *Clip* success rates for different 3B models trained with GRPO on the `GSM8K` training data (**left**) and evaluated on the `GSM8K` test set (**right**). Smoothed values are shown in full opacity, with original curves in the background. *TROLL* generally causes models to pick up a training signal more quickly, and exhibits more stable training behavior.

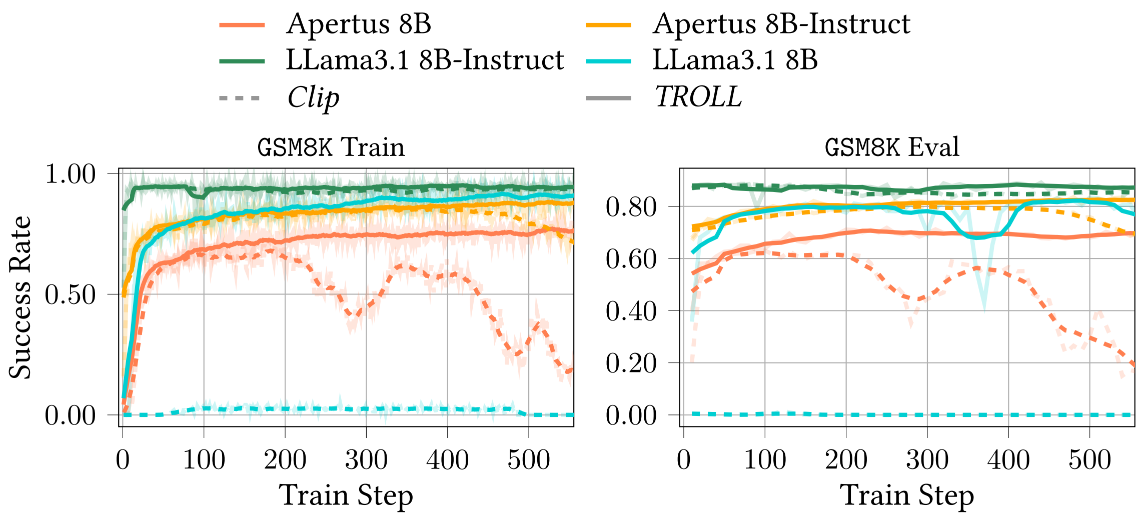


Figure D.3.: *TROLL* and *Clip* success rates for different 8B models trained with GRPO on the `GSM8K` training data (**left**) and evaluated on the `GSM8K` test set (**right**). Smoothed values are shown in full opacity, with original curves in the background. *TROLL* enables stable training for both Apertus and Llama3.1, whereas *Clip* leads to divergence in both cases.

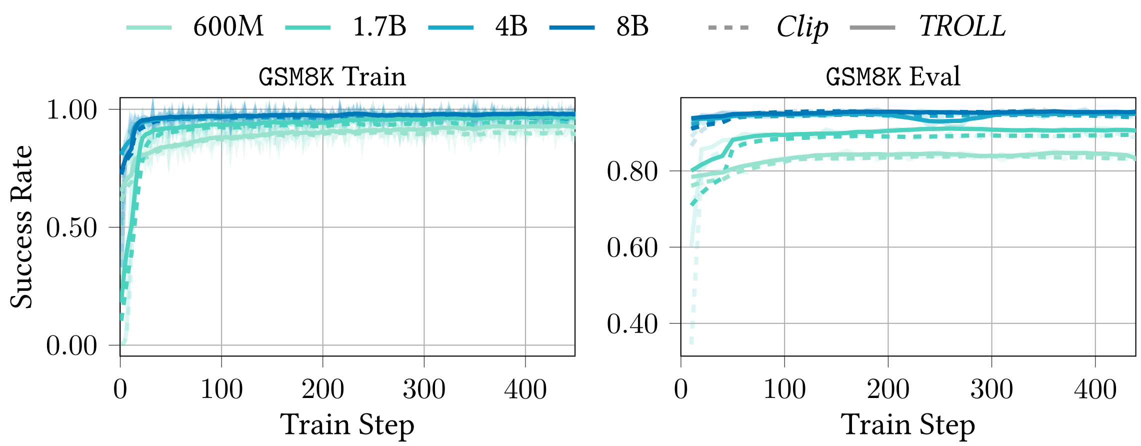


Figure D.4.: *TROLL* and *Clip* success rates for Qwen3 models with 600M to 8B parameters trained with GRPO on the `GSM8K` training data (**left**) and evaluated on the `GSM8K` test set (**right**). Smoothed values are shown in full opacity, with original curves in the background. Both *TROLL* and *Clip* quickly converge in all cases, although *TROLL* achieves slightly higher performance for most model sizes.

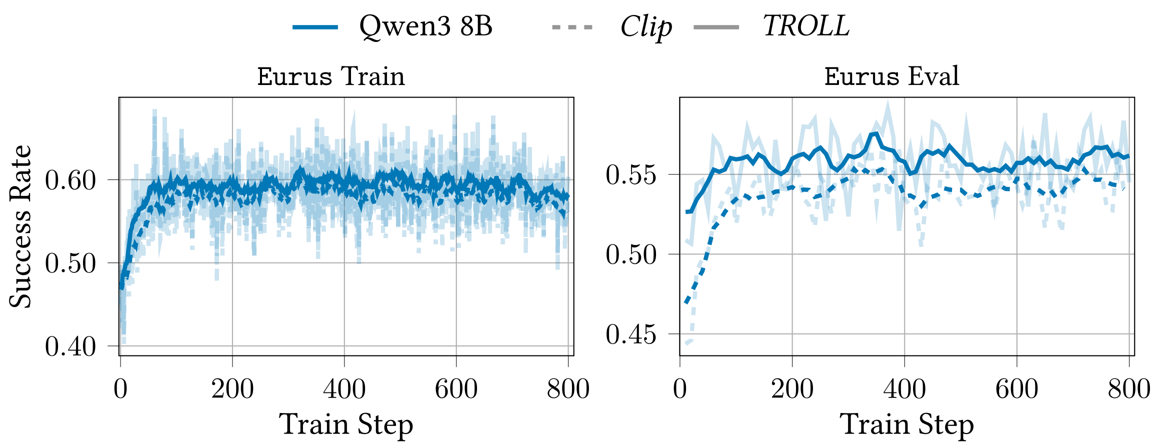


Figure D.5.: *TROLL* and *Clip* success rates for Qwen3-8B trained with GRPO on the `Eurus-Math` training data (**left**) and evaluated on the `Eurus-Math` test set (**right**). Smoothed values are shown in full opacity, with original curves in the background. *TROLL* performs slightly but consistently better during training, and generalizes well to the test set.

D.5. Example Generations

Prompt 1

```
system
Your task is to follow a systematic, thorough reasoning process before providing the final solution. This involves analyzing, summarizing, exploring, reassessing, and refining your thought process through multiple iterations. Structure your response into two sections: Thought and Solution. In the Thought section, present your reasoning using the format: "<think>
 {thoughts} </think>
". Each thought should include detailed analysis, brainstorming, verification, and refinement of ideas. After "</think>
," in the Solution section, provide the final, logical, and accurate answer, clearly derived from the exploration in the Thought section.
user
A list of positive integers has the following properties:
$\bullet$ The sum of the items in the list is $30$.
$\bullet$ The unique mode of the list is $9$.
$\bullet$ The median of the list is a positive integer that does not appear in the list itself.
 Find the sum of the squares of all the items in the list.
Present the answer in LaTeX format: \boxed{Your answer}.
assistant
```

***TROLL* 14B Response**

<think> </think>
Thought: Okay, let's try to figure this out. The list has positive integers. Sum is 30. Unique mode is 9, so 9 appears more times than any other number, and no other number appears the same number of times as 9. The median is a positive integer not in the list. Let me think about the median.
Median is the middle value when sorted. If the number of elements is odd, median is the middle element. If even, it's the average of two middle numbers. But the median is a positive integer not in the list. So if the number of elements is odd, the median is an element of the list, which contradicts. Therefore, the number of elements must be even. Let me denote n as the number of elements, even. So n is even, say n=2k. Then median is average of k-th and (k+1)-th elements. Since median is integer, so the average of those two elements is integer, so they are both integers and their sum is even. So they are both even or both odd.

Also, mode is 9, so 9 appears at least twice. And no other number appears that many times.
Let me try to find possible lists.
Let me try with n=6 elements. Then median is average of 3rd and 4th elements.
Suppose elements sorted: a, b, c, d, e, f. Median is (c + d)/2, which is integer, and not in the list. So c + d is even, and (c + d)/2 is not equal to any element.
Sum a + b + c + d + e + f = 30.
Mode is 9, so 9 appears more than any other number. Let's try with two 9s. If two 9s, then other numbers can appear at most once. Let's try.
If two 9s. Let me try to have two 9s. Let me assume elements are [x, y, z, w, v, u], sorted. Suppose two 9s. Let me try positions. If two 9s, they could be in positions 1 and 2, but then median is (c + d)/2. If two 9s, maybe c and d are something else.
Wait, if there are two 9s, and others are unique. Let me try an example.
Suppose list is [1, 2, 3, 4, 9, 20]. Sum is 1+2+3+4+9+20=39 too big.
Need sum 30. Maybe more numbers. Maybe more elements. Let me try n=6.
If two 9s, then other numbers are distinct. So total numbers: 9,9,a,b,c,d. All a,b,c,d distinct and different from 9.
Sum: 9+9+a+b+c+d=18 + a + b + c + d = 30 $\Rightarrow$ a + b + c + d = 12.
Also, sorted list: a, b, c, d, 9, 9? No, depends on values. Wait, sorted list would be [a, b, c, d, 9, 9] if a $\leq$ b $\leq$ c $\leq$ d $\leq$9. But median is (c + d)/2. Which is integer, and not in the list.
If a, b, c, d are distinct and different from 9. Let me try numbers.
Possible a, b, c, d: positive integers, distinct, not 9, sum to 12.
Possible set: 1, 2, 3, 6 (sum 12). Check if they are distinct and not 9. Yes. Then sorted list: [1,2,3,6,9,9]. Median is (3 + 6)/2 = 4.5, not integer. Not good.
Another set: 1, 2, 4, 5 (sum 12). Sorted list: [1,2,4,5,9,9]. Median is (4 +5)/2=4.5. Still not integer.
Another set: 1, 3, 4, 4. But duplicates. Not allowed since mode is unique 9. So can't have two 4s.
Another set: 2, 2, 2, 6. Again duplicates. Not allowed.
Another set: 1, 1, 5, 5. Duplicates again.
How about three 9s? If three 9s, then other numbers can appear at most two times.
Let me try n=7 elements. Wait, but median would be 4th element, which is in the list. But median is not in list. So n must be even. So n=6, 8, etc.
Try n=8 elements. Then median is average of 4th and 5th elements. Let me try with three 9s. So numbers: 9,9,9,a,b,c,d,e. Sum is 27 + a + b + c + d + e = 30 $\Rightarrow$ a + b + c + d + e = 3. Since positive integers, they are 1,1,1,0,... but 0 not allowed. So impossible.
n=6 elements, three 9s: sum would be 27 + others. If n=6, three 9s, others sum to 3. Only possible if other numbers are 1,1,1. But then mode is 9 and 1, which contradicts unique mode. So invalid.
So back to n=6, two 9s. Maybe the median is integer. For example, if c and d are both even or both odd. For example, if c=3, d=5: median 4, which is not in list. If

list is [1, 2, 3, 5, 9, 10]. Sum is 1+2+3+5+9+10=30. Mode is 1,2,3,5,9,10 all appear once. No mode. Not good.
Wait, need mode 9. So need more 9s. Maybe four 9s. n=7 elements: median is 4th element. Which would be 9, but median is not in list. Contradiction. n=8 elements: median is average of 4th and 5th elements. Let me try n=8. Four 9s. Others can appear at most three times. Sum: 4*9=36 >30. Too big.
n=6 elements, two 9s. Maybe other numbers have duplicates but less than two. Wait, no, if two 9s, others can appear once. So previous attempt.
Wait, what if there are two 9s, and two of another number? But mode is unique, so no. So all others appear once.
Wait, maybe n=5 elements. Median is 3rd element, which is in list. But problem says median not in list. So n must be even.
Maybe n=4 elements. Median is average of 2nd and 3rd elements. Let's try n=4. Two 9s. Sum is 9+9+a+b=18 +a +b=30 $\Rightarrow$a +b=12. a and b are positive integers, sorted: a $\leq$ b $\leq$9 or something. If sorted list: [a, b, 9, 9]. Median is (b +9)/2. Must be integer, so b +9 even $\Rightarrow$b odd. Also median not in list. So (b +9)/2 $\neq$ a, b, 9.
Let me try b=5, then median is (5+9)/2=7. 7 not in list. a=12 -5=7. So list is [7,5,9,9]. But sorted: [5,7,9,9]. Median is (7+9)/2=8, which is not in list. Okay. Mode is 9. Sum is 5+7+9+9=30. Perfect!
Wait, sum is 5+7+9+9=30. Yes. Median is 8, which is not in list. Mode is 9, unique. All conditions satisfied.
So the list is [5,7,9,9]. Sum of squares: 25 + 49 + 81 + 81 = 25 + 49 = 74, 74 + 81 = 155, 155 +81= $\boxed{236}$.
Wait, but wait: the median is (7 +9)/2=8, which is integer and not in list. Mode is 9, which appears twice, others once. Perfect.
So answer is $\boxed{236}$.
Wait, but let me check if there are other possible lists. For example, if a=3, b=9, but then b=9, a=3. List sorted [3,9,9,9]. Mode is 9, median is (9+9)/2=9, which is in list. Invalid.
Another case: a=5, b=7 as above. Another possibility: a=1, b=11. But b=11 >9, but sorted list [1,11,9,9] is invalid, need to sort: [1,9,9,11]. Median is (9 +9)/2=9, which is in list. Invalid.
a=3, b=9: sorted [3,9,9,9], median 9 invalid.
a=5, b=7: valid.
a=7, b=5: same as above.
a=2, b=10: sorted [2,9,9,10]. median (9+9)/2=9 invalid.
a=4, b=8: sorted [4,8,9,9]. median (8+9)/2=8.5 not integer.
a=6, b=6: sorted [6,6,9,9]. mode is 6 and 9, invalid.
Thus only possible list is [5,7,9,9], sum of squares $\boxed{236}$.
</think>
Thought: After checking all possibilities, the only valid list is [5,7,9,9], which satisfies all conditions.
Solution: The sum of the squares of all the items in the list is $\boxed{236}$.

Clip 14B Response

<think> </think>
Thought: Okay, let's see. I need to find a list of positive integers where the sum is 30, the unique mode is 9, and the median is a positive integer that doesn't appear in the list. Then, I have to find the sum of the squares of all the items.
First, let's recall what each term means. The mode is the number that appears most frequently. Since it's the unique mode, 9 must appear more times than any other number. So, there has to be at least two 9s, and no other number can appear that many times.
The median is the middle number when the list is ordered. If the list has an odd number of elements, the median is the middle one. If even, it's the average of the two middle numbers. But the problem says the median is a positive integer that doesn't appear in the list. So, if the list has an even number of elements, the average of the two middle numbers must be an integer not in the list. If odd, the middle number must not be in the list. But since the median is a positive integer, and the list contains positive integers, that's possible.
Let me try to figure out the possible lengths of the list. Let's denote the length of the list as n.
If n is odd, say n = 2k + 1, then the median is the (k+1)th element. This median must not be in the list. But if the median is not in the list, then the (k+1)th element is some number not present in the list. But all elements are positive integers, so the median must be an integer. But how can the median not be in the list? For example, if the list is [1, 2, 3, 4, 5], the median is 3, which is in the list. But if the list is [1, 2, 4, 5], the median is (2+4)/2 = 3, which is not in the list. So, for even n, the median can be an integer not in the list.
So, maybe the list has an even number of elements. Let's assume that. Let's try n = 4. Then the median is the average of the second and third elements. Let's say the list is [a, b, c, d], sorted. Then median is (b + c)/2. This must be an integer not in the list.
But also, the sum is 30. And the mode is 9. So, 9 must appear at least twice. Let's try to construct such a list.
If n = 4, and mode is 9, then at least two 9s. Let's say two 9s. Then the other two numbers must be different from each other and from 9, or one of them could be same as another but not more than once. But since mode is unique, no other number can appear twice.
So, for example, [x, 9, 9, y], sorted. Then the median is (9 + 9)/2 = 9. But 9 is in the list, which violates the condition that the median is not in the list. So that's no good.
So, if we have two 9s, and the median is not 9, then the two middle numbers must not both be 9. So, maybe the list is [a, 9, 9, b], but sorted. If a and b are such that the median is not 9. But if a <= 9 <= b, then the median is (9 + 9)/2 = 9. So that's not possible. So, maybe we need more than two 9s?

If we have three 9s. Let's try n = 5. Then the median is the third element. If the list is [a, b, 9, c, d], sorted. Then the median is 9. But 9 is in the list, which violates the condition. So that's no good.
If n = 6. Then the median is the average of the third and fourth elements. Let's say the list is [a, b, x, y, c, d], sorted. Then median is (x + y)/2. This must be an integer not in the list.
Also, mode is 9. So, 9 must appear more times than any other number. Let's say 9 appears three times. Then other numbers can appear at most two times.
Let me try to construct such a list. Let's say the list has three 9s. Let's arrange them in the list. For example, [a, b, 9, 9, 9, d], sorted. Then the third and fourth elements are 9 and 9. So the median is (9 + 9)/2 = 9, which is in the list. Not allowed.
So, maybe the three 9s are not in the middle. Let's say the list is [a, 9, 9, 9, b, c], sorted. Then the third and fourth elements are 9 and 9. Again, median is 9. Not allowed.
So, maybe the three 9s are spread out. For example, [a, 9, b, 9, c, 9], sorted. Then the third and fourth elements are b and 9. So the median is (b + 9)/2. This must be an integer not in the list.
Let me pick b such that (b + 9) is even. So, b must be odd. Let's say b = 7. Then median is (7 + 9)/2 = 8. So 8 is the median, and it's not in the list. That works.
So, the list is [a, 9, 7, 9, c, 9], sorted. Wait, but sorted order would be [a, 7, 9, 9, c, 9]. But that's not sorted unless a <= 7 and c >= 9.
Let me make it sorted. Let's say a <= 7 <= 9 <= c <= 9. So, a <= 7, and c >= 9. But since there are three 9s, maybe c is 9. So the list is [a, 7, 9, 9, 9, 9]. But then the third and fourth elements are 9 and 9. Median is 9, which is in the list. Not allowed.
Hmm. So, maybe the three 9s are not consecutive. Let me try [a, 7, 9, 9, c, d], sorted. Then the third and fourth elements are 9 and 9. Median is 9. Not allowed.
So, maybe we need four 9s. Let's try that. If there are four 9s, then the mode is 9, and no other number appears more than three times.
Let's try n = 7. Then the median is the fourth element. If the list is [a, b, c, 9, d, e, f], sorted. Then the median is 9, which is in the list. Not allowed.
If n = 8. Then the median is the average of the fourth and fifth elements. Let's say the list is [a, b, c, x, y, z, w, v], sorted. Then median is (x + y)/2. This must be an integer not in the list.
If we have four 9s. Let's say the list is [a, b, c, 9, 9, 9, 9, d], sorted. Then the fourth and fifth elements are 9 and 9. Median is 9, which is in the list. Not allowed.
So, maybe the four 9s are spread out. For example, [a, b, 9, c, d, 9, e, 9], sorted. Then the fourth and fifth elements are c and d. So, median is (c + d)/2. Let's pick c and d such that this is an integer not in the list.
Let me try c = 7 and d = 9. Then median is (7 + 9)/2 = 8. Not in the list. So that works.
So the list is [a, b, 9, 7, 9, 9, e, 9], sorted. Wait, but sorted order would be [a, b, 7, 9, 9, 9, e, 9]. So, a and b must be <= 7, and e >= 9.

Let me pick a and b as small as possible. Let's say a = 1, b = 2. Then the list is [1, 2, 7, 9, 9, 9, e, 9]. Sorted, it would be [1, 2, 7, 9, 9, 9, e, 9]. But e must be >= 9. Let's say e = 9. Then the list is [1, 2, 7, 9, 9, 9, 9, 9]. But then the fourth and fifth elements are 9 and 9. Median is 9, which is in the list. Not allowed.
So, e must be greater than 9. Let's say e = 10. Then the list is [1, 2, 7, 9, 9, 9, 10, 9]. Sorted: [1, 2, 7, 9, 9, 9, 9, 10]. Now, the fourth and fifth elements are 9 and 9. Median is 9, which is in the list. Not allowed.
So, maybe c and d are not 7 and 9. Let's try c = 7 and d = 10. Then median is (7 + 10)/2 = 8.5, not an integer. Not allowed.
c = 6 and d = 10. Median is (6 + 10)/2 = 8. Good. So, the list is [a, b, 6, 9, 10, 9, e, 9], sorted. Wait, sorted would be [a, b, 6, 9, 9, 9, e, 10]. So, a and b <= 6, e >= 9.
Let me pick a = 1, b = 2. Then the list is [1, 2, 6, 9, 9, 9, e, 10]. Sorted: [1, 2, 6, 9, 9, 9, e, 10]. The fourth and fifth elements are 9 and 9. Median is 9, which is in the list. Not allowed.
So, maybe the fourth and fifth elements are not both 9. Let me try to have the four 9s spread out more. For example, [a, b, 9, c, d, 9, e, 9], sorted. Then the fourth and fifth elements are c and d. Let me pick c = 7 and d = 8. Then median is (7 + 8)/2 = 7.5, not integer.
c = 7 and d = 9. Median is 8. So, the list is [a, b, 9, 7, 9, 9, e, 9], sorted: [a, b, 7, 9, 9, 9, e, 9]. Again, fourth and fifth are 9 and 9. Median is 9.
This seems tricky. Maybe I need to have more than four 9s? Let's try five 9s.
If n = 9. Then the median is the fifth element. If the list is [a, b, c, d, 9, e, f, g, h], sorted. Then the median is 9, which is in the list. Not allowed.
If n = 10. Then the median is the average of the fifth and sixth elements. Let's say the list has five 9s. So, [a, b, c, d, 9, 9, 9, 9, 9, e], sorted. Then the fifth and sixth elements are 9 and 9. Median is 9, which is in the list. Not allowed.
So, maybe the five 9s are spread out. For example, [a, b, c, 9, d, e, f, 9, g, 9], sorted. Then the fifth and sixth elements are d and e. Let me pick d = 7 and e = 8. Then median is (7 + 8)/2 = 7.5, not integer.
d = 7 and e = 9. Median is (7 + 9)/2 = 8. Not in the list. So that works.
So the list is [a, b, c, 9, 7, 9, f, 9, g, 9], sorted. Wait, sorted would be [a, b, c, 7, 9, 9, f, 9, g, 9]. So, a, b, c <= 7, f >= 9, g >= 9.
Let me pick a = 1, b = 2, c = 3. Then the list is [1, 2, 3, 7, 9, 9, f, 9, g, 9]. Sorted: [1, 2, 3, 7, 9, 9, f, 9, g, 9]. The fifth and sixth elements are 9 and 9. Median is 9, which is in the list. Not allowed.
So, maybe the fifth and sixth elements are not both 9. Let me try to have the five 9s spread out more. For example, [a, b, 9, c, d, e, f, 9, g, 9], sorted. Then the fifth and sixth elements are d and e. Let me pick d = 7 and e = 8. Then median is (7 + 8)/2 = 7.5, not integer.
d = 7 and e = 9. Median is 8. So, the list is [a, b, 9, c, 7, 9, f, 9, g, 9], sorted: [a, b, 7, 9, 9, 9, f, 9, g, 9]. The fifth and sixth elements are 9 and 9. Median is 9.
This is really challenging. Maybe I need to go back to n = 6 and try again.

Let me try n = 6, with three 9s. The median is (third + fourth)/2. Let me have the list [a, b, 7, 9, c, d], sorted. Then the third and fourth elements are 7 and 9. Median is (7 + 9)/2 = 8. Not in the list. That works!
So, the list is [a, b, 7, 9, c, d], sorted. With three 9s. So, one of a, b, c, d is 9. Let's say c and d are 9. So, the list is [a, b, 7, 9, 9, 9], sorted. Then a and b <= 7.
Let me pick a = 1, b = 2. Then the list is [1, 2, 7, 9, 9, 9]. Sum is 1 + 2 + 7 + 9 + 9 + 9 = 37. Too much.
We need sum to be 30. So, let's adjust. Let me try a = 1, b = 1. Then the list is [1, 1, 7, 9, 9, 9]. Sum is 1 + 1 + 7 + 9 + 9 + 9 = 36. Still too much.
a = 1, b = 2, and change 7 to something smaller. But 7 is the third element. If I make it 6, then the list is [1, 2, 6, 9, 9, 9]. Sum is 1 + 2 + 6 + 9 + 9 + 9 = 36. Still too much.
Maybe a = 1, b = 1, and third element is 5. Then list is [1, 1, 5, 9, 9, 9]. Sum is 1 + 1 + 5 + 9 + 9 + 9 = 34. Still too much.
a = 1, b = 1, third = 4. Sum is 1 + 1 + 4 + 9 + 9 + 9 = 33.
a = 1, b = 1, third = 3. Sum is 1 + 1 + 3 + 9 + 9 + 9 = 32.
a = 1, b = 1, third = 2. Sum is 1 + 1 + 2 + 9 + 9 + 9 = 31.
a = 1, b = 1, third = 1. Sum is 1 + 1 + 1 + 9 + 9 + 9 = 30. Perfect!
So the list is [1, 1, 1, 9, 9, 9]. Sorted. Let's check the conditions:
- Sum is 1 + 1 + 1 + 9 + 9 + 9 = 30. Check. - Mode is 9, which appears three times. Other numbers appear once or twice. So unique mode is 9. Check. - Median is (third + fourth)/2 = (1 + 9)/2 = 10/2 = 5. 5 is not in the list. Check.
So this list satisfies all conditions!
Now, the sum of the squares is $1^2 + 1^2 + 1^2 + 9^2 + 9^2 + 9^2$ = 1 + 1 + 1 + 81 + 81 + 81 = 3 + 243 = 246.
So the answer is $\boxed{246}$.
But wait, let me check if there are other possible lists. For example, could there be a list with more elements?
But given the constraints, this seems to be the only possible list. So I think

Declaration of Generative AI Usage

The following generative AI tools were used in the preparation of this dissertation.

1. Technical Specifications

Tool/System	Model Version	Provider
Google Gemini	Gemini 3.0 Pro	Google
Claude	Opus 4.6	Anthropic

2. Scope and Nature of Application

The systems listed above were used as linguistic and stylistic aids. Their application was limited to the following tasks:

- **Paragraph Optimization:** Refining the structural coherence and clarity of individual paragraphs without altering the underlying scientific arguments.
- **Refining Transitions and Flow:** Improving the narrative transitions between sections to ensure a logical and cohesive narrative.
- **Grammatical and Syntactic Review:** Systematic verification of the manuscript for spelling, grammar, and technical punctuation.

3. Statement of Responsibility

I hereby declare that:

- All original research, core hypotheses, data analysis, and technical conclusions are the sole intellectual property of the author.
- No scientific content, primary data, or independent ideas were generated by the AI systems.

- Every AI-suggested revision was manually reviewed, critically appraised, and adapted by the author to ensure factual accuracy.
- The author remains fully responsible for the integrity and accuracy of the final text.