Versatile Manipulation with Generative Imitation Learning from Robot Play Data

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte
Dissertation

von
Moritz Benno Reuss
aus Bad Friedrichshall

Tag der mündlichen Prüfung: 09.06.2026
1. Referent: TT-Prof. Dr. Rudolf Lioutikov
2. Referent: Prof. Dr. Edward Johns

© 2026 Moritz Benno Reuss

This document is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0):
https://creativecommons.org/licenses/by/4.0/

Third-party material is excluded from this license where indicated; the rights of the respective copyright holders apply.

Abstract

General-purpose robots capable of performing diverse manipulation tasks in unstructured environments remain a long-standing goal in robotics. While traditional imitation learning has succeeded in controlled settings, scaling these methods to the open world requires learning from large, uncurated, and heterogeneous datasets of "play" data. However, effective learning from such data presents three fundamental challenges: the need to model highly multimodal behavior distributions, the difficulty of learning from sparse language annotations, and the computationally prohibitive cost of scaling current policies towards foundation behavior models.

This thesis proposes novel data-driven methods to address these challenges by combining expressive generative modeling with parameter-efficient architectures and self-supervised learning. First, we introduce BESO, a novel policy representation using score-based diffusion policy that captures the multimodality of play data. We show that BESO outperforms prior policy representations in expressivity and learning multimodal behavior. Second, to address data with sparse language labels, we present MDT, a transformer-based diffusion policy that leverages two self-supervised objectives to learn from partially labeled datasets. By aligning latent goal-conditioned embeddings and introducing masked visual foresight, MDT enables robust language-following behaviors despite sparse supervision.

Building on these expressive representations, we address the computational bottleneck of scaling diffusion policies. We introduce MoDE, a Mixture-of-Experts diffusion policy that utilizes a novel noise-conditioned routing mechanism to scale model capacity without increasing inference latency. Finally, we unify these contributions in FLOWER, a compact Vision-Language-Action (VLA) model. By employing intermediate-modality fusion and global action-space conditioning, FLOWER achieves state-of-the-art performance across 190 tasks in 10 benchmarks. It matches the performance of 7B-parameter models with less than 1B parameters and requires only 1% of the pretraining compute. Together, these contributions demonstrate that combining

generative expressivity with architectural efficiency is key to democratizing generalist robot policies.

Zusammenfassung

Universell einsetzbare Roboter, die diverse Manipulationsaufgaben in unstrukturierten Umgebungen ausführen können, bleiben ein langjähriges Ziel der Robotik. Während traditionelles Imitationslernen in kontrollierten Umgebungen erfolgreich war, erfordert die Skalierung dieser Methoden für die offene Welt das Lernen aus großen, unkuratierten und heterogenen Datensätzen von "Play-Daten". Effektives Lernen aus solchen Daten stellt jedoch drei fundamentale Herausforderungen: die Notwendigkeit, hochgradig multimodale Verhaltensverteilungen zu modellieren, die Schwierigkeit, aus wenigen Sprachannotationen zu lernen, und die hohen Rechenkosten bei der Skalierung aktueller Robotik-Policies hin zu Foundation-Modellen für generelles Roboterverhalten.

Diese Arbeit schlägt neuartige datengetriebene Methoden vor, um diese Herausforderungen zu bewältigen, indem sie expressive generative Modellierung mit parametereffizienten Architekturen und selbstüberwachtem Lernen kombiniert. Erstens führen wir BESO ein, eine neuartige Policy-Repräsentation basierend auf score-basierten Diffusionsmodellen, welche die Multimodalität von Play-Daten erfasst. Wir zeigen, dass BESO frühere Policy-Repräsentationen hinsichtlich Expressivität und dem Erlernen multimodalen Verhaltens übertrifft. Zweitens präsentieren wir MDT, eine Transformer-basierte Diffusions-Policy, die das Problem fehlender Sprachannotationen adressiert. Durch die Nutzung zweier selbstüberwachter Lernziele, der Ausrichtung latenter, zielkonditionierter Modellrepräsentationen und einer maskierten visuellen Zukunftsvorhersage (Masked Visual Foresight), erlernt MDT robuste sprachgesteuerte Verhaltensweisen trotz nur teilweise vorhandener sprachlicher Annotationen.

Aufbauend auf diesen expressiven Repräsentationen adressieren wir die rechnerischen Limitierungen bei der Skalierung von Diffusions-Policies. Wir führen MoDE ein, eine Mixture-of-Experts-Diffusions-Policy, die einen neuartigen rauschkonditionierten Routing-Mechanismus nutzt, um die Modellkapazität zu skalieren, ohne die Inferenzlatenz zu erhöhen. Schließlich vereinen

wir alle diese Beiträge in FLOWER, einem kompakten Vision-Language-Action-Modell (VLA). Durch den Einsatz von Intermediate-Modality-Fusion und globaler Aktionsraum-Konditionierung erzielt FLOWER State-of-the-Art-Leistung in 190 Aufgaben über 10 Benchmarks hinweg. Es erreicht die Leistung von 7B-Parameter-Modellen mit weniger als einer Milliarde Parametern und benötigt lediglich 1% des Rechenaufwands für das Pretraining.

Zusammenfassend zeigen diese Beiträge, dass die Kombination aus generativer Expressivität und architektonischer Effizienz der Schlüssel zur Demokratisierung generalistischer Roboter-Policies ist.

Acknowledgements

First and foremost, I want to thank my supervisor and mentor, Rudi, for your unwavering support throughout my PhD journey. Your relentless energy and optimism in critical situations pushed me beyond my limits during critical moments like paper sprints before the deadlines. I am grateful for sharing your passion for research, and for the fun evenings at conferences with karaoke and beers. It was a fun ride to see the lab grow and move our robots a few times.

I owe special thanks to Christine for being the heart of our group, keeping everything running smoothly, and tracking the many administrative details I routinely forget. I also thank Edward Johns for agreeing to serve on my thesis committee.

To all my colleagues at the Intuitive Robots Lab: thank you for creating such a positive and collaborative work environment. Special recognition goes to Max and Hongyi, my earliest colleagues, as well as to my former students turned colleagues: Nils and Paul.

I am deeply grateful to my former master's thesis supervisors, Philipp and Vaisakh, who helped me decide to pursue a PhD and provided essential guidance as I learned the fundamentals of research. I still remember Philipp mentioning that I would make a good PhD student, one of the most meaningful compliments I have received as a researcher. I want to thank all my colleagues at the Autonomous Learning Robots (ALR) Lab for creating such a welcoming office environment and for the many engaging conversations about research and life, particularly in the early stages of my PhD when your senior advice proved invaluable. Special thanks go to my former office mates Denis, David, and Woo-Jong for the humor and discussions that made coming to the office consistently enjoyable.

I extend particular thanks to my collaborator and friend Jyo for the rewarding experience of working on the MoDE project together, as well as for many

insightful conversations about research and personal stuff. Thanks also to Nikos for countless discussions about good and bad robot policies, the challenges of reviewer 2, and all the other small things that shape a PhD experience. I am also grateful to my external collaborators Oier, Manthan, and Jonas for enjoyable collaborations that resulted in exciting papers and broadened my perspective on research. Special thanks to my former students Ömer, Marcel, and Fabian, who all went several extra miles, with nights of little sleep and many real robot rollouts before paper deadlines to make MDT and FLOWER possible. I also thank my Apple internship supervisors, Peide, Walter, and Jian, for a formative research and personal experience in California, and also a big thanks to Apple for sponsoring the final part of my PhD research.

A heartfelt thank you to my wife, Annika, for her unwavering support and patience through many deadline sprints. Her encouragement has meant more than she knows.

Finally, I thank my parents, Thomas and Sonja, for providing me with every opportunity and supporting me at each step of my life. Their unconditional belief in everything I pursue with passion made the decision to enter research an easy one.

And lastly, a nod to the snowstorm of January 2023 in Boston, which gave me the unexpected gift of an extra day to polish and submit the BESO paper to RSS.

Contents

Abstract i

Zusammenfassung iii

Acknowledgements v

List of Figures xiii

List of Tables xxi

1. Introduction 1
1.1. Chapter Overview and Contributions 2
1.2. Published Papers 5

2. Preliminaries 7
2.1. Deep Generative Models 7
2.1.1. Energy-based Models 9
2.1.2. Score-based Generative Models 10
2.1.3. Continuous Time Score-based Diffusion Models . . . 12
2.1.4. Flow Matching 14
2.1.5. Classifier-Free Guidance 15
2.2. Transformers and Foundation Models 15
2.2.1. Attention Mechanism 16
2.2.2. Transformer Experts 17
2.2.3. Transformer Norms 18
2.2.4. Position Embeddings 19
2.2.5. Vision Transformer (ViT) 21
2.2.6. CLIP: Contrastive Language-Image Pretraining . . . 22
2.2.7. Mixture of Experts (MoE) 23
2.2.8. Large-Language Models (LLMs) 25
2.2.9. Vision-Language Model (VLMs) 26

2.2.10. Vision-Language-Action Models (VLAs) 27
2.3. Imitation Learning 28
2.3.1. Behavior Cloning (BC) 28
2.3.2. Goal-Conditioned Imitation Learning (GCIL) 30
2.3.3. Learning from Play (LfP) 30
2.3.4. Action representations 31
2.3.5. Simulation Benchmarks 33

3. Goal-conditioned Imitation Learning using Score-based Diffusion Policies 41
3.1. Introduction 42
3.2. Related Work 44
3.3. Method 46
3.3.1. Score-based Diffusion Policies 46
3.3.2. Diffusion Training 47
3.3.3. Efficient Action Generation using Deterministic Samplers 48
3.3.4. Goal-Guided Score-based Diffusion Policies 49
3.3.5. Diffusion Wrapper 50
3.3.6. Model Architecture 50
3.4. Evaluation 52
3.4.1. Comparative Evaluation 53
3.4.2. Ablations 62
3.4.3. Limitations 66
3.5. Conclusion 67

4. Multimodal Diffusion Transformer: Learning Versatile Behavior from Multimodal Goals 69
4.1. Introduction 69
4.2. Related Work 71
4.3. Method 74
4.3.1. Problem Formulation 74
4.3.2. Score-based Diffusion Policy 74
4.3.3. Model Architecture 75
4.3.4. Masked Generative Foresight 76
4.3.5. Aligning Latent Goal-Conditioned Representations . 77
4.4. Evaluation 79
4.4.1. Evaluation on CALVIN 80
4.4.2. Evaluation on LIBERO 82

4.4.3. Real Robot Experiments . . . 83
4.4.4. Evaluation of Auxiliary Losses . . . 86
4.4.5. Additional Ablation Studies . . . 87
4.4.6. Limitations . . . 90
4.5. Conclusion . . . 90

5. Efficient Diffusion Transformer Policies with Mixture of Expert Denoisers for Multitask Learning . . . 93
5.1. Introduction . . . 94
5.2. Related Work . . . 95
5.3. Method . . . 96
5.3.1. Problem Formulation . . . 97
5.3.2. Diffusion Policy . . . 97
5.3.3. Mixture-of-Experts Denoising . . . 97
5.3.4. Generalist Policy Pre-Training . . . 100
5.4. Evaluation . . . 101
5.4.1. State-based Experiments . . . 101
5.4.2. Generalist Policy Evaluation . . . 103
5.4.3. Long-Horizon Multi-Task Experiments . . . 104
5.4.4. Scaling Multi-Task Experiments . . . 105
5.4.5. Generalization Experiments . . . 107
5.4.6. Computational Efficiency of MoDE . . . 108
5.4.7. Ablation Studies . . . 108
5.4.8. Limitations . . . 113
5.5. Conclusion . . . 113

6. FLOWER: Democratizing Generalist Robot Policies with Efficient Vision-Language-Flow Models . . . 115
6.1. Introduction . . . 116
6.2. Related Work . . . 117
6.3. Method . . . 120
6.3.1. Intermediate Modality Fusion Vision-Language-Action-Models . . . 120
6.3.2. Cross-Action Space Flow Transformer . . . 121
6.3.3. Rectified Flow for Action Generation . . . 122
6.3.4. FLOWER: Efficient Flow-based Vision-Language-Action Models . . . 123

6.4. Evaluation . . . 123
6.4.1. Evaluation of Critical Design Decisions for Efficient Flow VLAs . . . 124
6.4.2. Simulation Experiments . . . 126
6.4.3. Real-World Evaluation and Generalization . . . 129
6.5. Limitations . . . 131
6.6. Conclusion . . . 131

7. Conclusion . . . 133
7.1. Summary . . . 133
7.2. Discussion and Outlook . . . 135
7.2.1. Follow-Up Research Directions . . . 135
7.2.2. Fundamental Open Research Problems in Imitation Learning . . . 136
7.2.3. Closing Perspective . . . 137

Bibliography . . . 139

A. Supplementary Material for BESO . . . 159
A.1. BESO Hyperparameters . . . 159
A.2. BESO Baselines . . . 160
A.3. Additional Experiments . . . 162
A.3.1. BESO Rollout Examples and Failure Cases . . . 162

B. Supplementary Material for MDT . . . 165
B.1. MDT Hyperparameters . . . 165
B.2. MDT Baselines . . . 165
B.3. MDT Additional Ablations . . . 166
B.3.1. MDT Example Rollouts . . . 169
B.3.2. MDT Real Robot Results . . . 169

C. Supplementary Material for MoDE . . . 173
C.1. MoDE Hyperparameters . . . 173
C.2. MoDE Baselines . . . 173
C.3. MoDE Mixture-of-Experts Ablations . . . 174

D. Supplementary Material for FLOWER . . . 181
D.1. FLOWER Hyperparameters . . . 181
D.2. FLOWER Model and Pretraining Details . . . 181
D.2.1. Pretraining Ablation Experiments . . . 182

D.2.2. Language Prompt for the VLA 184
D.2.3. Custom Learning Rate Scheduler 184
D.2.4. Details for Cross-Action Space Flow Transformer . . 185
D.3. FLOWER Detailed Experiments 185
D.3.1. CALVIN Benchmark 186
D.3.2. SIMPLER Benchmark 187
D.3.3. LIBERO Benchmark 188
D.3.4. ALOHA Benchmark 189
D.3.5. Real Kitchen Play Dataset 190
D.3.6. Generalization Experiments 193

List of Figures

1.1. **Overview of the research questions and contributions of the thesis.** 3

2.1. **Overview of core Deep Generative Models and their mathematical relationships. Top-Left:** Energy-Based Models (EBMs) define probability via an energy function $E_\theta(\mathbf{x})$ but suffer from an intractable partition function Z_θ. **Top-Right:** Score-Based Models (NCSM) bypass Z_θ by learning the score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ via denoising score matching across multiple discrete noise levels. **Bottom-Left:** Diffusion Models generalize score matching to continuous time $t \in [0, T]$ using Stochastic Differential Equations (SDEs). This framework serves as the foundation for the BESO, MDT, and MoDE methods discussed in this thesis. **Bottom-Right:** Rectified Flow Models simplify the generative process by learning a velocity field v_θ that follows straight paths between data and noise, serving as the foundation for FLOWER. 8

2.2. **Detailed architecture of the Transformer model**. **Right:** the full Transformer consisting of L layers. **Middle:** the internal structure of a single residual Transformer block including normalization and MLP layers. **Left:** The specific operations within the Self-Attention mechanism. The left panel explicitly highlights the projection of Queries, Keys, and Values, along with optional components (dashed lines) such as RMSNorm and RoPE. 16

2.3. **Architecture of a Sparse MoE Transformer block**. Following Self-Attention, a learned Router $\mathcal{G}(\mathbf{z})$ activates only the top-k experts (here $k = 2$) via gates w_1, w_3. The weighted sum aggregates outputs from selected experts (solid borders), while inactive experts (dashed) are skipped to scale capacity without increasing computation. 24

2.4. **Relay Kitchen example rollouts.** Two example rollouts in the Relay Kitchen benchmark introduced by Gupta et al. [57], illustrating different task compositions. 32
2.5. **Block Push example rollouts.** Two rollouts in the multimodal Block Push benchmark based on Implicit Behavioral Cloning [47, 172], showing different block-to-target assignments and execution orders. 34
2.6. **Overview of the CALVIN environment.** (a) CALVIN contains four different settings (A,B,C,D) with different configurations of slides, drawers and textures. (b) Example rollout consisting of 5 tasks in sequence. The next goal is only given to the policy if it completes the prior goal. The environment images and rollout illustrate the CALVIN benchmark introduced by Mees et al. [130]. 34
2.7. **Overview of the LIBERO benchmark.** Representative tasks from the Goal, Object, Spatial, and LIBERO-100 suites. Images adapted from Figures 8–11 of Liu et al. [110] (cropped, rearranged, and rescaled). The LIBERO-100 montage illustrates tasks used in the 90/Long split. 36
2.8. **SIMPLER task overview.** Simulation frames adapted from the supplementary videos accompanying Li et al. [106]. The left four images show the WidowX/BridgeData V2 tasks; the right four show Google Robot tasks. 37
2.9. **Overview of the real robot play kitchen setup.** The left image shows the play kitchen with its relevant objects in the scene, while the right image shows the cameras and the second robot used for teleoperation. The policies are conditioned on the side-view camera and a birds-eye view camera. 38
2.10. **Overview of the 20 tasks recorded in our real world kitchen.** Both camera perspectives used for training policies are recorded. 39

3.1. **Overview of the action generation process of BESO with the used model architecture. Left:** General Action Generation Process using the Diffusion Process to denoise the next action given the current observation $\boldsymbol{s}$ and desired goal state $\boldsymbol{g}$ in N-steps. **Middle:** the high-level score model with its preconditioning layers and skip connections. **Right:** the internal denoising score model, which uses a transformer architecture with causal masking to iteratively predict the denoised action given the sequence of prior observations, actions, and the goal sequence. 42
3.2. **Visualization of the Seq-BESO model architecture.** Seq-BESO predicts a sequence of future actions conditioned on the current state token and desired goal with causal masking. 51
3.3. **Overview of the Aligning task for testing the multimodality of different policies.** The robot is tasked with pushing the box to a randomly generated target position. The dataset contains 1000 human recorded demonstrations with 500 trajectories for each behavior. 53
3.4. **Ablation study on dataset size: performance on the Aligning task when training on different fractions of the dataset.** BESO outperforms all other policy representations in all dataset sizes in success rate and entropy of learned solutions. In addition, BESO has the lowest standard deviations. 57
3.5. **Performance on the CALVIN image-only benchmark.** Left: average success rate on 13 individual tasks. Right: success rate when conditioned on a single goal image requiring completion of 2 sequential tasks. 58
3.6. **Visualization of Seq-BESO rollouts in the play kitchen.** The rollouts demonstrate **BE**havior Generation using **S**c**O**re-based Diffusion models (BESO)'s capability to handle complex manipulation tasks involving articulated objects and requiring precise actions. BESO can efficiently perform a diverse set of tasks with only a few demonstrations. 60

3.8. **Comparison of CFG Method for Goal-Conditioned Imitation Learning (GCIL) from Play Data.** For CFG-BESO we evaluate the 10 seeds on 100 rollouts each with different λ values. The CFG variant of BESO has a slightly worse average result in both environments with similar rewards. When using $\lambda = 0$, we can recover an unconditional policy, that performs random rollouts with high rewards and low results. 63

4.1. **Overview of MDT.** (Left) Overview of the proposed multimodal Transformer-Encoder-Decoder Diffusion Policy used in Multimodal Diffusion Transformer (MDT). (Right) Specialized Diffusion Transformer Block for the Denoising of the Action Sequence. MDT learns a goal-conditioned latent state representation from multiple image observations and multimodal goals. The camera images are processed either via frozen Voltron Encoders with a Perceiver or ResNets. The separate GPT denoising module iteratively denoises an action sequence of 10 steps with a Transformer Decoder with Causal Attention. It consists of several Denoising Blocks, as visualized on the right side. These blocks process noisy action tokens with self-attention and fuse the conditioning information from the latent state representation via cross-attention. MDT applies adaLN conditioning [148] to condition the blocks on the current noise level. In addition, it aligns the latent representation tokens of the same state with different goal specifications using self-supervised contrastive learning. The latent representation tokens are also used as a context input for the masked Image Decoder module to reconstruct masked-out patches from future images. 71

4.2. **Visualization of the Masked Generative Foresight Auxiliary Task of MDT.** It starts by encoding the current observation and goal using the MDT Encoder. The resulting latent state representations then serve as conditional inputs for the Future Image-Decoder. This decoder receives encoded patches of future camera images along with mask tokens. Its task is to reconstruct the occluded patches in future frames. 75

4.3. **Visualization of Contrastive Latent Alignment (CLA).** During training, we enforce same observation with different goal modalities to be close in the multimodal latent space of the policy, while negative examples from other states with other goals are pushed away using InfoNCE loss objective. 78
4.4. **Overview of the different environments used to test MDT**. (Left) CALVIN Benchmark consisting of four environments each with unique positions and textures for slider, drawer, LED, and lightbulb. (Middle) Overview of the different tasks and scene diversity in the LIBERO benchmark, which is divided into 5 different task suites. (Right) Example tasks from the real robot experiments at a toy kitchen, where models are tested after training on partially labeled play data. 80
4.5. **Study on the performance of Masked Generative Foresight and the Contrastive Latent Alignment for MDT.** We analyse the impact of both auxiliary tasks on the ABCD CALVIN challenge. The results show the average rollout length over 1000 instruction chains averaged over 3 seeds. 85
4.6. **Study on the performance of our proposed Masked Generative Foresight (MGF) and Contrastive Latent Alignment (CLA) objectives for pretraining on action-free data.** We pretrain MDT on LIBERO-90 with the objectives and test the average performance on all LIBERO-Long tasks with different number of demonstrations. The results show the success rate averaged over 20 rollouts for all 10 tasks and 3 seeds. LfS refers to trained from scratch and PrT are all pretrained models. 88

5.1. **Overview of MoDE.** Left: transformer with noise-conditional self-attention and noise-conditioned routing. Right: the router activates subsets of MLP experts based on noise level during denoising. 94
5.2. **Visualization of Router Caching.** After training MoDE, the router is noise-conditioned, allowing pre-computation of the experts used at each noise level before inference. This enables removing the router and retaining only the selected experts, significantly improving network efficiency. 99

5.3. **Visualization and Results for LIBERO environment.** (a) Example tasks from the LIBERO benchmark of Liu et al. [110]. (b) Our evaluation results on LIBERO-10 and LIBERO-90, averaged over 3 seeds with 20 rollouts for each task. 104
5.4. **Computational efficiency comparison between Mixture-of-Denoising Experts Policy (MoDE) and a Dense-Transformer model with the same number of parameters.** Left: Average inference speed over 100 forward passes for various batch sizes. Right: FLOPs count for MoDE with router cache and without compared against a dense baseline. MoDE demonstrates superior efficiency with lower FLOPs count and faster inference thanks to its router caching and sparse expert design. 108
5.5. **Visualized Expert Utilization.** The average usage of all experts in MoDE across all layers is shown. Purple color corresponds to low usage and yellow color to high one, and each image is separately normalized. The average activation shows that MoDE learns to utilize different experts for different noise levels. 111

6.1. **Intermediate fusion for efficient VLA policies.** Our fusion strategy (top-right) strategically prunes VLM layers while enhancing Flow Transformer capacity in parameter-constrained settings. This approach informs FLOWER, a novel, 950M VLA that achieves competitive performance across 10 benchmarks using only 1% of the pretraining compute of models like OpenVLA [89], while maintaining a small memory footprint across diverse embodiments and action spaces (bottom). 116
6.2. **Visualization of the Florence With Embodied Flow (FLOWER) architecture.** A fine-tuned VLM processes multimodal inputs and integrates intermediate features into a Flow Transformer via cross-attention. The model predicts velocity fields with Global AdaLN-Zero conditioning for the action space, embodiment, and temporal metadata. 120
6.3. **Visualization of AdaLN for Diffusion/Flow Policies.** Comparison of standard DiT blocks and our proposed Global AdaLN with layer-specific Lora adapters. 122

6.4. **Simulation Environments used to test FLOWER.** From left to right: *CALVIN* [130], *LIBERO* [110], *SIMPLER* [106] with the *Bridge* and *Google Robot* variants and *Aloha Simulation Benchmark* [222]. The simulation panels credit the original benchmark papers; the ALOHA Cube Transfer and Bimanual Insertion tasks were introduced with ACT [222]. The final two panels show our own real-world kitchen and generalization settings with clutter, different lighting, and novel objects. 124
6.5. **Real World Results for Multi-Task and Generalization Experiments** 128
6.6. **Overview of Real World Tasks and Generalization Settings**. The left half shows example tasks used to evaluate policies in the play kitchen setup. The right part shows different challenges of generalization. We test policies in cluttered environments, under reduced lighting and with novel objects. 129
6.7. **Generalization Experiments:** Examples of unseen objects (left) and cluttered scenes (right) used to test the adaptability of the policies in our real-world setting. All the tested objects are not included in the training dataset. 129

A.1. **Visualization of BESO Rollouts in the Play Kitchen.** The sequences showcase successful policy rollouts from BESO for 7 tasks in a real-world play kitchen environment, demonstrating the model's effectiveness across diverse kitchen-related activities. 163

B.1. **Overview of two Diffusion Transformer Baseline Architectures for MDT.** The first variant uses a transformer encoder but also processes the noise as a token. The second one is the Transformer Diffusion policy presented in Chi et al. [27] without any encoder. 168
B.2. **Real Robot rollouts with goal image conditioning using MDT.** The first column shows the goal image used for the rollout. 4 out of 6 tasks are successful. The robot fails to open the oven door and opens the ice box instead. 170
B.3. **Overview of the real robot kitchens setup.** The left image shows the play kitchen with all its objects, while the right image shows the cameras and second robot used for data collection with our robot used for teleoperation. 170

C.1. **Scaling performance of MoDE and Dense-MoDE on CALVIN ABC and ABCD environments.** The plots show average performance for 2 to 8 experts using best-performing variants for each environment. 175
C.2. **Average Expert Utilization for different Load Balancing Weights across all denoising levels for MoDE.** 176
C.3. **Average Expert Utilization for different number of used experts across all denoising levels.** 177
C.4. **Average Expert Utilization for two variants that use topk=1.** The left plot shows MoDE with 4 experts, the right shows MoDE with 4 experts, where 1 is used in all settings as a shared one. Overall, MoDE struggles to equally distribute tokens across experts in this setting. 178
C.5. **Visualization of used experts in pretrained MoDE.** This image shows the expert usage distribution across 12 layers in a Mixture of Experts (MoE) model. Each subplot represents a different layer (from layer 0 to layer 11), with experts labeled E0, E1, E2, and E3 on the x-axis. The y-axis indicates the log-scaled token count, displaying the frequency of tokens routed to each expert within that layer. The color gradient indicates the proportion of tokens assigned to each expert, where darker colors represent higher usage. 180

D.1. ***Aloha Simulation Tasks.*** Our average success rates over 500 evaluations on the *Cube Transfer* and *Bimanual Insertion* simulation tasks introduced by Zhao et al. [222]. S denotes FLOWER trained from scratch, X applies cross-action space pretraining, and $\mathcal{J}$ represents DROID-only joint state pretraining. 190
D.2. **Example generalization rollouts.** First two rows show rollouts with background distractors, rows 3 and 4 show rollouts with only a flashlight as a light source, and the last 3 rows showcase novel objects. 196

List of Tables

3.1. **Average performance of policies on the conditioned block-push and kitchen environment.** Results are averaged over 10 seeds with 100 rollouts each. Rew. denotes total reward for completing any task; Res. denotes completion of the specified goals. The latent-plan baseline is LMP. BESO and CFG-BESO consistently outperformed all baselines, despite using only 3 inference steps. Both variants of BESO show a low deviation across seeds, indicating their robustness. 52

3.2. **Evaluation of different policy representations on the Aligning Task.** We report the average success rate and a measurement of the Entropy, that measures the multimodality on different methods. BESO achieves the highest success rate and entropy. . 54

3.3. **Performance of Seq-BESO and baselines on the CALVIN benchmark.** We evaluate on three settings: single-environment training (D→D), zero-shot generalization (ABC→D), and large-scale multi-environment training (ABCD→D). All methods are tested on environment D following the protocol of [130]. Results are averaged over 1000 rollouts with instruction chains of 5 tasks and 3 random seeds. 59

3.4. **Evaluation results for play kitchen for Seq-BESO and GCBC.** The results are averaged over 10 rollouts each. 61

3.5. **Evaluation of the Influence of Noise Injection for the performance of BESO.** Results are averaged over 2 samplers with and without random noise injection. 65

3.6. **Ablation for different time steps for denoising of BESO.** Results are averaged over 10 seeds and 100 rollouts each. All models use the DDIM solver with 3 denoising steps. 66

4.1. **Performance comparison of various policies on the CALVIN ABCD→D and D→D benchmark.** The table shows the average rollout length to solve 5 instructions in a row (Avg. Len.) of 1000 chains. The numerical results are reproduced from Table I of the final RSS publication [163], including its reported ± values. MDT and MDT-V outperform the baselines included in that comparison. The source's 0.00 entry for RoboFlamingo is retained as printed; its text states that no seed standard deviation was available for this baseline. 81

4.2. **Overview of the performance of MDT and baselines with and without our proposed Self-Supervised Losses on LIBERO Task suites.** All results show the average performance of all tasks averaged over 20 rollouts each and with 3 seeds. MDT outperforms Transformer-BC in several settings with only 2% of language annotations. Lang Ann. refers to Language Annotations. All numerical entries, including the reported Average and ± values, are reproduced verbatim from Table II of the final RSS publication [163]. 82

4.3. **Single-task results for MDT and Baselines in the play kitchen with language goals.** The average is computed over 5 rollouts for each of the 20 tasks. The poor performance of MT-ACT showcases the difficulty of this experiment. MDT performs significantly better with an additional boost through the auxiliary objectives. 84

4.4. **Average rollout length achieved by each method on the long-horizon real robot play kitchen setup.** The performance is averaged over 4 instruction chains with 4 rollouts each. MDT clearly outperforms MT-ACT. The performance of MDT is further increased substantially by the auxiliary CLA and MGF objectives. The relatively short avg rollout lengths emphasize how challenging this setting is, even for strong state-of-the-art methods such as MT-ACT. 86

5.1. Dataset sampling weights used for training MoDE on a small subset of trajectories. The total dataset consists of 196k trajectories. 100

5.2. State-based goal-conditioned evaluation averaged over 4 seeds, reproduced from the MoDE study [160]. Relay Kitchen reports the mean number of requested goals completed (maximum 4); Block Push reports ordered-task success. These are separate evaluations from the 10-seed results in Chapter 3. 102
5.3. **SIMPLER evaluation on Google Robot tasks.** MoDE is pretrained on the OXE mixture and then finetuned on Fractal; baseline cells retain the reported comparison results. Success is full task completion. Each Avg. is the unweighted mean of the three displayed task categories, and Overall is the mean of all six displayed cells. The four-category Google Robot comparison in Chapter 6 additionally includes apple placement and uses the FLOWER evaluation setting. 103
5.4. **SIMPLER evaluation on WidowX tasks.** Grasp measures partial completion; Success measures full task completion. Full Success Average is the unweighted mean of the four Success columns. Mixed Average preserves the earlier aggregate over all eight Grasp and Success columns, as reported in the MoDE study [160]; it is not a full-task success rate. 103
5.5. **Performance comparison on the two CALVIN challenges.** The table reports average success rates for individual tasks within instruction chains and the average rollout length (Avg. Len.) to complete 5 consecutive instructions, based on 1000 chains. Zero standard deviation indicates methods without reported standard deviations. "Prt" denotes models requiring policy pretraining. Parameter counts exclude language encoders. 106
5.6. **Ablation Studies for MoDE on LIBERO-10.** All results are averaged over 3 seeds with 20 rollouts each. 109
5.7. **Overview of the performance of all different token routing strategies used for MoDE across 5 benchmarks.** We mark the best result for each environment in **bold** and the second best in *italics*. We use CAL to represent CALVIN. To average the results, we normalize all scores and compute the average over all environments. Only the CALVIN ABC ablation results use a separate, weaker training run than the main ABC benchmark. . 110
5.8. **Performance comparison of MoDE with standard vs Flow Matching objective on CALVIN ABC→D.** MoDE-FM achieves comparable performance with just 2 denoising steps. 112

6.1. Evaluation of VLA fusion strategies. 125
6.2. **Comparing Layer Pruning** for CALVIN ABC and LIBERO Long. 125
6.3. **Ablation results for FLOWER on CALVIN ABC.** 125
6.4. Comparison of FLOWER across simulation benchmarks. 127
6.5. **Mean inference efficiency** (1000 steps in Bf16). All policies except OpenVLA use chunk length 50. *LF* refers to Late Fusion Ablation of FLOWER with the complete VLM. 130

A.1. **Overview of the most important Hyperparameters for BESO for tested environments.** Img-C refers to image-only CALVIN experiments and Lang-C refers to Language-conditioned CALVIN benchmark experiments. 159

B.1. **Summary of all the Hyperparameters for the MDT policy used in the CALVIN experiments and the ones of Multi-Task Action Chunking Transformer (MT-ACT).** 166
B.2. **Overview of the chosen hyperparameters for our Image Demasking Model used in the MGF loss, that uses a vision transformer architecture.** 167
B.3. **Overview of the hyperparameters for Distill-D on the CALVIN and LIBERO benchmark.** Our code is based on the Diffusion-policy implementation from Chi et al. [27] with our continuous-time diffusion variant. To guarantee a fair comparison the hyperparameters for Distill-D and MDT regarding diffusion are the same. 167
B.4. **Ablation on the best prediction horizon for Masked Generative Foresight.** Results are reported on CALVIN ABCD→D with MDT-V and LIBERO-Spatial with MDT. 168
B.5. **Evaluation of the Performance Increase of the MT-ACT policy with the additional Masked Generative Foresight Loss on the CALVIN ABCD→D challenge.** 168

B.6. **Overview of the performance influence of MGF and Contrastive Alignment on MDT-V on the CALVIN ABCD→D challenge.** In addition, the performance of both transformer ablations are also shown. Moreover, results for MDT with ResNets as image goal encoders are reported with and without auxiliary objectives. The results are reported over 1000 rollouts averaged over 3 seeds. Numerical entries, including reported averages and uncertainties, are reproduced from Table IX of the final RSS publication [163]; they have not been recomputed from the displayed chain probabilities. 169
B.7. **Ablation on different Masking Rates for Masked Generative Foresight.** Results are reported on CALVIN D→D with MDT-V and on LIBERO-Spatial with MDT. 169
B.8. **Detailed results of our real robot kitchen experiments for single-task setting with language goals of MDT.** All results are averaged over 5 rollouts. "right" and "left" are abbreviated with "rt" and "lt". 171
B.9. **Detailed results of our long-horizon, real robot multi-task experiments for MDT and baselines.** 172

C.1. **Summary of all the Hyperparameters for the MoDE policy used in our experiments.** 173

D.1. **Overview of Hyperparameters used for FLOWER across different Benchmarks** 181
D.2. **Overview of Parameter Distribution across all Model Components of FLOWER.** 182
D.3. **Hyperparameters for Pretraining FLOWER on all Pretraining Data Mixtures** 183
D.4. **Dataset Distribution for Pretraining of FLOWER.** Results are reported by percentage 183
D.5. CALVIN benchmark results for D, ABC and ABCD. 186
D.6. **Experimental Results for the SIMPLER Bridge Benchmark.** Average Performance comparison across all Tasks in the Bridge Setting. 187

D.7. **Experimental Results for the SIMPLER Google Robot Benchmark.** Reported full-task success for the four-category FLOWER comparison. The shared baseline entries coincide with the Visual Matching values in Chapter 5; they are not equal-weight averages of Visual Matching and Variant Aggregation. Average summarizes the four displayed categories, including apple placement. 187
D.8. **Experimental Results for the LIBERO Benchmarks.** SR: Success Rate. Best results in each column are shown in bold. FLOWER leads the Object and Long suites and remains competitive across the other settings. 189
D.9. **Detailed Results for all tested Real Robot Tasks in the Kitchen Environment.** Each task has two rows: the first (SR) reports success rate (%), and the second (R) reports rank within that task (lower rank = better performance). The best results per task are highlighted in **bold**. 195
D.10. **Generalization experimental results for novel objects, distractions and new lighting conditions.** The table reports the success rate (in %) of the corresponding policy evaluated under three different generalization scenarios: Novel Object, Flashlight, and Background Distractors (evaluated 3 times for each setting). The best score for each test is highlighted in **bold**. A dash (-) indicates that the task was not evaluated in that scenario. 197
D.11. **Long Horizon Task Composition Results.** For each sequence task, the per-instruction success rates (in %) are shown for the first 5 instructions (if applicable) along with the average sequence length. "–" indicates that no instruction was successfully solved at that index. The Overall Performance rows report the average success rate (computed over all available instructions) and the average sequence length across tasks for each method. 197

List of Acronyms

AR Autoregressive Models

BC Behavior Cloning

BESO **BE**havior Generation using **S**c**O**re-based Diffusion models

CFG Classifier-Free Guidance

CLA Contrastive Latent Alignment

CLIP Contrastive Language-Image Pretraining

CNN Convolutional Neural Network

DDPM Denoising Diffusion Probabilistic Model

DGM Deep Generative Model

DSM Denoising Score Matching

EBM Energy-based Model

EDM **E**lucidating the **D**esign Space of Diffusion-Based Generative **M**odels

FLOWER **Flo**rence **W**ith **E**mbodied Flow

GAIL Generative Adversarial Imitation Learning

GAN Generative Adversarial Network

GCIL Goal-Conditioned Imitation Learning

IL Imitation Learning

IRL Inverse Reinforcement Learning

LfP Learning from Play

LLM Large-Language-Model

MDT Multimodal Diffusion Transformer

MGF Masked Generative Foresight

MLP Multi-Layer Perceptron

MoDE **M**ixture-of-**D**enoising **E**xperts Policy

MoE Mixture-of-Experts

MSE Mean Squared Error

MT-ACT Multi-Task Action Chunking Transformer

MTIL Multi-Task Imitation Learning

ODE Ordinary Differential Equation

OXE Open-X-Embodiment

RF Rectified Flow

RL Reinforcement Learning

RMS Root-Mean-Squared

RoPE Rotary Position Embedding

SDE Stochastic Differential Equation

SDM Score-based Diffusion Model

SoTA State-of-The-Art

VAE Variational Autoencoder

ViT Vision Transformer

VLA Vision-Language-Action-Model

VLM Vision-Language-Model

1. Introduction

Imagine commanding a robot: *"The dishwasher is finished, can you empty it?"* and watching it open the door, retrieve each item, and place it in the correct cabinet. This capability of understanding free-form language, perceiving cluttered environments and executing long-horizon sequences of precise manipulations, would transform manufacturing, surgery, and elderly care. Yet, despite decades of robot progress, it remains an unsolved challenge. Not because robots lack the hardware for the required dexterity, but because we lack methods for teaching robots to acquire these skills autonomously.

The fundamental barrier to achieving these general-purpose robots is versatile manipulation. The pre-programmed and rigid methods used in industry are brittle; they fail immediately when faced with novel objects or unseen setups. Learning-based methods are essential to achieving this goal. The most commonly used learning-based paradigms are Reinforcement Learning (RL) [192] and Imitation Learning (IL) [169]. IL [169] is particularly promising: it bypasses the exploration and sample-efficiency problems of RL by learning directly from expert demonstrations. Crucially, its reliance on demonstration data positions it to benefit from the large-scale data that have revolutionized other fields.

Learning from large-scale data has resulted in breakthroughs in image generation [166] and text generation [194]. This scaling of data has yielded remarkable capabilities on challenging tasks like in-context reasoning [205, 204], mathematical problem solving [175], and synthesizing realistic images or videos from text [13, 201]. This success suggests a path forward for robot learning: data-centric approaches can achieve similar emergent capabilities for versatile manipulation policies.

However, robotics faces a unique constraint: the *robot data gap*. Collecting data for text and images is easy and scalable, whereas generating high-quality robot data is difficult [7]. Only a few large-scale robot datasets exist, and every local robot workstation differs in camera placement, lighting, and

general layout. Furthermore, different robots prefer different action spaces and frequencies [222, 28], and collecting demonstrations requires resets after every task completion. This bottleneck renders traditional IL approaches inapplicable and requires a more efficient data collection paradigm. Human skill acquisition offers a useful analogy.

Consider how humans learn manipulation: A child does not practice grasping thousands of red mugs in the same tabletop setup before generalizing to a green mug. Instead, children learn dexterous manipulation by playfully interacting with diverse objects and transferring this knowledge across categories. The standard IL data collection paradigm is the opposite: narrow, unimodal task repetition rather than behavioral diversity. Overcoming this limitation and scaling IL towards versatile robots requires better data collection paradigms.

Learning from Play (LfP) [119, 130] offers a solution to these challenges. LfP instructs data collectors to play freely with all available objects, rather than focusing on a single task. This task-agnostic scheme streamlines data collection by eliminating manual environment resets, which accelerates the acquisition of diverse behavioral data. Recent open-source efforts such as Open-X-Embodiment (OXE) [30] and DROID [85] used this approach to provide diverse, cross-embodiment data.

However, LfP introduces new challenges for the policies. Play data is highly multimodal (humans solve tasks in many ways) and contains long-horizon behaviors without clear segmentation. LfP is faster and easier to collect, but necessary language annotations require substantial manual work with post-hoc annotation and segmentation. Thus, existing LfP datasets often contain only partial labels for learning language-conditioned behaviors [117, 130]. To build generalist agents from play, we require novel policy architectures that are expressive, data-efficient, and computationally tractable.

1.1. Chapter Overview and Contributions

This thesis addresses three fundamental challenges:

C1: Policy Expressivity for Multimodal Behavior. The first challenge is finding a policy representation that can learn effectively from diverse play data. Human demonstrations contain many valid ways to solve a task, yet

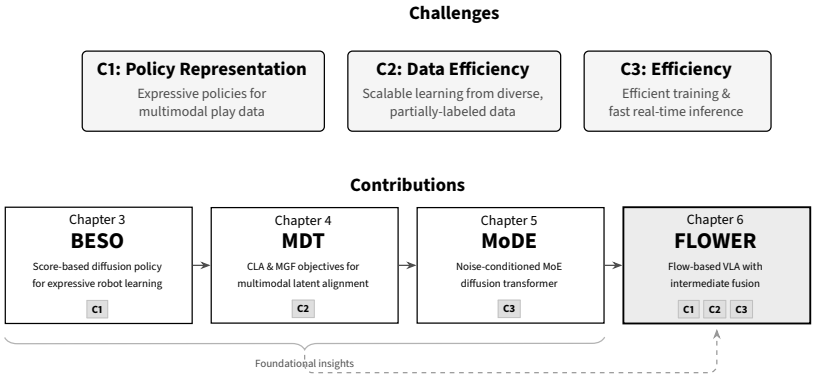


Figure 1.1.: Overview of the research questions and contributions of the thesis.

existing policy representations fail to capture this diversity: continuous heads average out behaviors, discrete tokenizers lack precision, and Energy-based Models (EBMs) [97] can be difficult to train [47].

C2: Learning from Heterogeneous, Partially-Labeled Play Data. The second major challenge concerns data efficiency. Scaling policies requires data from diverse embodiments and action spaces. It cannot rely solely on expensive, fully-annotated language labels to learn free-form instruction following. This necessitates methods that can efficiently learn from sparse language labels by also leveraging other available modalities, such as goal images, to achieve robust instruction following. In addition, efficient learning from different robot embodiments requires architectures that can handle heterogeneous data at scale.

C3: Computational Efficiency for Training and Deployment. The final challenge is practicality. Current state-of-the-art foundation models in robotics are computationally demanding (high FLOPs and parameters) for widespread access and real-time training and deployment. Established Vision-Language-Action-Models (VLAs) like OpenVLA [89] have 7B+ parameters and require several thousand hours of GPU training. This creates significant entry barriers for many researchers and necessitates research towards more efficient foundation models that achieve similar results with less compute. In addition, inference of foundation models for robotics is often limited by real-time constraints for dynamic manipulation. Therefore, we need methods that

enable the benefits of data-driven scaling while running fast on commodity hardware.

Chapter 2 establishes the theoretical background and introduces the benchmarks used for evaluation. The following chapters present our main contributions:

Chapter 3 addresses **C1** by introducing BESO, a score-based diffusion policy. At the start of this research (2022–2023), existing policy representations for IL did not meet all requirements for learning from uncurated play data. Discrete action representations lack expressiveness, while continuous heads are not multimodal. BESO solves this by learning the score function of the action distribution. It surpasses all prior policies in expressivity and scaling, achieving state-of-the-art results across all data regimes. We further ablate crucial diffusion design decisions to enable fast inference in just three denoising steps.

Chapter 4 addresses **C2** with MDT. While BESO established the diffusion backbone, it relied on unimodal goal definitions and cannot handle sparse language labels. MDT overcomes this limitation by aligning latent goal-conditioned representations across modalities through two auxiliary objectives: CLA, which uses contrastive learning to align goal-conditioned state representations, and MGF, which reconstructs masked future frames via a small auxiliary network. We evaluate these objectives across 184 tasks (164 simulated and 20 real-world tasks) and find improvements on several benchmarks without increasing inference cost.

With expressive and multimodal goal-following policies established, the primary bottleneck becomes computational efficiency (**C3**). **Chapter 5** introduces MoDE, the first Mixture-of-Experts (MoE) diffusion policy, featuring a novel noise-conditioned routing strategy that learns specialist experts for different noise ranges. This design enables efficient expert caching and faster inference than dense transformers while using only half the active parameters. Pretrained on a large-scale robot-agnostic play dataset with limited compute, MoDE achieves State-of-The-Art (SoTA) performance on LIBERO [110] and CALVIN [130].

Finally, **Chapter 6** presents FLOWER, a compact, generalist VLA policy. Building upon our previous chapters, FLOWER tackles all proposed challenges (**C1–C3**) in a unified and efficient design. We introduce intermediate fusion to bridge the gap between pretrained Vision-Language-Models (VLMs) and

flow-based action generation. Pretrained on a cross-embodiment play dataset in just 200 GPU hours, FLOWER achieves SoTA performance across 190 tasks in 10 benchmarks. Our efficient design democratizes the training and usage of VLAs for a wide range of researchers and users.

Chapter 7 summarizes the contributions from previous chapters and discusses remaining limitations and future directions for data-driven imitation learning.

1.2. Published Papers

Below we provide an overview of all published and peer-reviewed papers that are the foundation for the contributions for this thesis:

- **Chapter 3 (BESO):** *Goal Conditioned Imitation Learning using Score-based Diffusion Policies* [162]. **Moritz Reuss**, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. In *Robotics: Science and Systems (RSS)*, 2023.
- **Chapter 4 (MDT):** *Multimodal Diffusion Transformer: Learning Versatile Behavior from Multimodal Goals* [163]. **Moritz Reuss**, Ömer Erdinç Yağmurlu, Fabian Wenzel, and Rudolf Lioutikov. In *Robotics: Science and Systems (RSS)*, 2024.
- **Chapter 5 (MoDE):** *Efficient Diffusion Transformer Policies with Mixture of Expert Denoisers for Multitask Learning* [160]. **Moritz Reuss**, Jyothish Pari, Pulkit Agrawal, and Rudolf Lioutikov. In *International Conference on Learning Representations (ICLR)*, 2025.
- **Chapter 6 (FLOWER):** *FLOWER: Democratizing Generalist Robot Policies with Efficient Vision-Language-Flow Models* [161]. **Moritz Reuss**, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Otto, and Rudolf Lioutikov. In *9th Annual Conference on Robot Learning (CoRL)*, 2025.

The following publications, which we co-authored during our dissertation, explore related topics such as novel action representations and 3D perception but are not presented as main chapters:

- *NaviTrace: Evaluating Embodied Navigation of Vision-Language Models* [207]. Tim Windecker, Manthan Patel, **Moritz Reuss**, Richard Schwarzkopf, Cesar Cadena, Rudolf Lioutikov, Marco Hutter, and Jonas Frey. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026.
- *BEAST: Efficient Tokenization of B-Splines Encoded Action Sequences for Imitation Learning* [224]. Hongyi Zhou, Weiran Liao, Xi Huang, Yucheng Tang, Fabian Otto, Xiaogang Jia, Xinkai Jiang, Simon Hilber, Ge Li, Qian Wang, Ömer Erdinç Yağmurlu, Nils Blank, **Moritz Reuss**, and Rudolf Lioutikov. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2025.
- *PointMapPolicy: Structured Point Cloud Processing for Multi-Modal Imitation Learning* [77]. Xiaogang Jia, Qian Wang, Anrui Wang, Han A. Wang, Balázs Gyenes, Emiliyan Gospodinov, Xinkai Jiang, Ge Li, Hongyi Zhou, Weiran Liao, Xi Huang, Maximilian Beck, **Moritz Reuss**, Rudolf Lioutikov, and Gerhard Neumann. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2025.
- *Scaling Robot Policy Learning via Zero-Shot Labeling with Foundation Models* [14]. Nils Blank, **Moritz Reuss**, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Wenzel, Oier Mees, and Rudolf Lioutikov. In *8th Annual Conference on Robot Learning (CoRL)*, 2024.
- *Towards Diverse Behaviors: A Benchmark for Imitation Learning with Human Demonstrations* [78]. Xiaogang Jia, Denis Blessing, Xinkai Jiang, **Moritz Reuss**, Atalay Donat, Rudolf Lioutikov, and Gerhard Neumann. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.
- *Information Maximizing Curriculum: A Curriculum-Based Approach for Learning Versatile Skills* [15]. Denis Blessing, Onur Celik, Xiaogang Jia, **Moritz Reuss**, Maximilian Xiling Li, Rudolf Lioutikov, and Gerhard Neumann. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

2. Preliminaries

This chapter covers core concepts in robot learning and generative models required to understand the contributions in the following chapters. We begin with generative models with an emphasis on score-based diffusion models (the basis for BESO, MDT and MoDE) and flow matching (the basis for FLOWER). Next, we cover the Transformer architecture, from its core concepts like attention to relevant foundation models like VLMs. Finally, we formalize imitation learning and describe the benchmarks used for evaluation.

This dissertation uses two temporal indices: i for discrete timesteps within a robot trajectory, and t for continuous time in the diffusion process. We define $\boldsymbol{a}_i$ as an action at timestep i of a trajectory and σ_t as the noise level at diffusion time t. Thus, $\boldsymbol{a}_{i,t}$ refers to a noisy version of action $\boldsymbol{a}_i$ at diffusion time t.

For clarity in later chapters, we often omit indices when context is unambiguous. Action chunks $\boldsymbol{a}_{i:i+H-1}$ and observation histories $\boldsymbol{s}_{i-h+1:i}$ are written simply as $\boldsymbol{a}$ and $\boldsymbol{s}$ when the horizon H and history length h are fixed within a section.

2.1. Deep Generative Models

Deep Generative Models (DGMs) are a class of neural networks trained to estimate the probability distribution over a high-dimensional dataset (e.g., videos, text or behaviors). By approximating this distribution during training, they are able to synthesize new samples from the data at test time. We define the data distribution as p_{data} and the model distribution as p_θ. Given a dataset, we optimize θ by gradient descent using a loss function that measures how different our learned p_θ is compared to p_{data}.

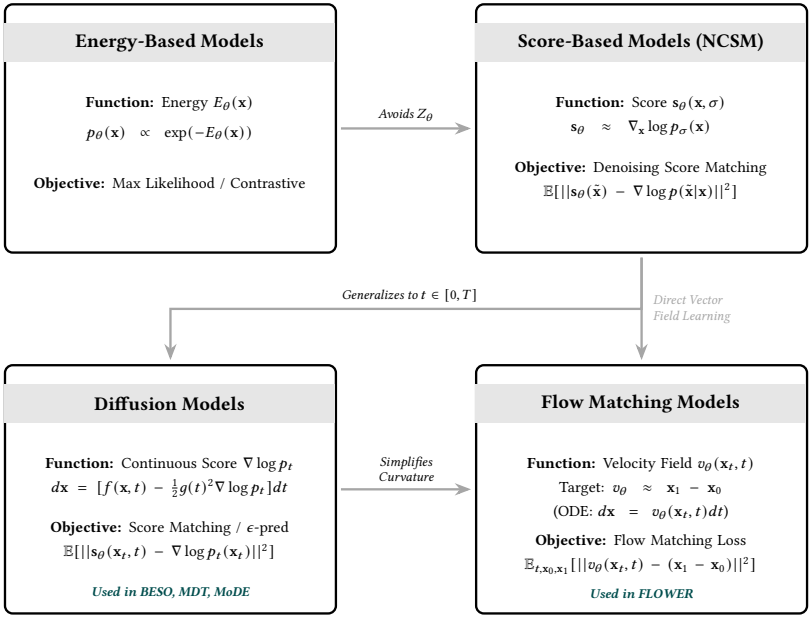


Figure 2.1.: Overview of core Deep Generative Models and their mathematical relationships. Top-Left: Energy-Based Models (EBMs) define probability via an energy function $E_\theta(\mathbf{x})$ but suffer from an intractable partition function Z_θ. **Top-Right:** Score-Based Models (NCSM) bypass Z_θ by learning the score function $\nabla_\mathbf{x} \log p(\mathbf{x})$ via denoising score matching across multiple discrete noise levels. **Bottom-Left:** Diffusion Models generalize score matching to continuous time $t \in [0, T]$ using Stochastic Differential Equations (SDEs). This framework serves as the foundation for the BESO, MDT, and MoDE methods discussed in this thesis. **Bottom-Right:** Rectified Flow Models simplify the generative process by learning a velocity field v_θ that follows straight paths between data and noise, serving as the foundation for FLOWER.

Figure 2.1 provides an overview of the core generative models discussed in this section and their relevance for the following chapters. The most influential ones in the literature are: Autoregressive Models (AR) [196, 94], Generative Adversarial Networks (GANs) [54], Diffusion Models [66, 188, 183], and Variational Autoencoders (VAEs) [91]. All of these families exhibit trade-offs between generation speed, tractability, and expressiveness.

The following subsections trace a conceptual path from energy-based models to the score-based diffusion and flow matching frameworks that underpin our methods. We begin with EBMs, which provide a flexible way to define probability densities but suffer from an intractable normalization constant. This limitation motivates score-based models, which sidestep normalization by learning the gradient of the log-density instead. We then show how diffusion models extend score matching to a continuous-time framework, and conclude with flow matching—a recently proposed alternative that learns straighter generative trajectories.

2.1.1. Energy-based Models

EBMs are a type of DGM which learn to define a probability density using an energy function $E_\theta(x)$. The energy function $E_\theta(x)$ assigns low energy to likely data configurations and high energy to others [97]. EBMs define a distribution by:

$$p_\theta(\mathbf{x}) = \frac{\exp(-E_\theta(\mathbf{x}))}{Z_\theta} \tag{2.1}$$

$$Z_\theta = \int_{\mathbb{R}^D} \exp(-E_\theta(\mathbf{x}))\, \mathrm{d}\mathbf{x} \tag{2.2}$$

where Z_θ is the *partition function*, which guarantees normalization of the learned probability density $\int_{\mathbb{R}^D} p_\theta(\mathbf{x})\, \mathrm{d}\mathbf{x} = 1$.

Training EBMs via maximum likelihood is challenging because the partition function, Z_θ, is intractable. This necessitates approximations, such as contrastive objectives relying on Monte Carlo methods (often referred to as negative sampling), which are frequently unstable and computationally expensive [47].

Given these challenges with training dynamics, another type of EBM training emerged which tries to avoid computing the normalizing constant: *score matching*. The score function of EBMs is independent of the partition function Z_θ and is used in many applications like Langevin Dynamics.

$$\nabla_x \log p_\theta(\mathbf{x}) = \nabla_x \left(\log \exp(-E_\theta(\mathbf{x})) - \log Z_\theta\right) \tag{2.3}$$

$$= \nabla_x \left(-E_\theta(\mathbf{x}) - \log Z_\theta\right) \tag{2.4}$$

$$= -\nabla_x E_\theta(\mathbf{x})\,. \tag{2.5}$$

This insight is the foundation for score-based generative models, which learn the score function directly instead of modeling the energy E_θ and estimating Z_θ.

2.1.2. Score-based Generative Models

Every density $p_{\text{data}}(\mathbf{x})$ has a corresponding score function, which is the gradient of the log-density that points towards regions with higher density:

$$\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}), \quad \text{where } \mathbf{s} : \mathbb{R}^D \to \mathbb{R}^D. \tag{2.6}$$

The primary advantage of modeling the score is that it bypasses the need to compute the intractable partition function Z_θ, as $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ is independent of this normalizing constant. This makes modeling the score equally expressive as modeling the density itself [93].

Score-based models are trained using the Fisher divergence objective [71]

$$\mathrm{E}_{p(\mathbf{x})}[||\nabla_{\mathbf{x}} \log p(\mathbf{x}) - s_\theta(\mathbf{x})||_2^2], \tag{2.7}$$

which compares the ℓ_2-distance between the data score and the learned score model. However, this objective is infeasible because it requires the unknown data score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$. We can use a class of objectives called *score matching* [71] which minimize the Fisher divergence without requiring the data score:

$$\mathcal{L}_{\text{SM}} = \mathrm{E}_{p(\mathbf{x})}[\mathrm{tr}(\nabla_{\mathbf{x}} s_\theta(\mathbf{x})) + \frac{1}{2}||s_\theta(\mathbf{x})||_2^2] \tag{2.8}$$

This objective shares the same optimal minimizer as the intractable Fisher Divergence objective, thereby achieving the same optimization goal. However, it cannot be scaled towards deep neural networks, because it requires the trace computation of the Jacobian, which scales quadratically with the dimension of $\mathbf{x}$ (D^2), becomes computationally prohibitive for high-dimensional data.

To avoid the expensive Jacobian trace computation, Denoising Score Matching (DSM) was introduced [199]. The explicit score matching objective is equivalent to training a model s_θ to do a simple denoising task. Given a sample of $\mathbf{x} \sim p_{\text{data}}$, we perturb $\mathbf{x}$ with a small amount of Gaussian noise $n \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathcal{I})$ to get $\tilde{\mathbf{x}}$. The updated Score Matching Objective becomes:

$$\mathcal{L}_{\text{DSM}} = \mathrm{E}_{p_{\text{data}}(\mathbf{x})\mathrm{E}_{p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}}[||s_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})||_2^2] \tag{2.9}$$

This updated target $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ is tractable for Gaussian Noise and simplifies to $-(\tilde{\mathbf{x}} - \mathbf{x})/\sigma^2$.

Once a score-based model is trained, we can generate new samples using Langevin dynamics:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \eta s_\theta(\mathbf{x}_n) + \sqrt{2\eta}\epsilon_n, \qquad \epsilon_n \sim \mathcal{N}(\mathbf{0}, \mathcal{I}) \tag{2.10}$$

in an iterative process starting at $\mathbf{x_0}$. For suitable regularity conditions, this converges to a sample from $p_{\text{data}}(\mathbf{x})$ for $K \to \infty$ iterations and $\eta \to \mathbf{0}$.

DSM introduces a general trade-off based on the choice of the noise level: Small perturbations do not cover low-density regions of the data, and corruption with larger noise provides good estimates of score for low-density regions while not being accurate for the high-density ones. This trade-off led Song and Ermon [185] to propose a more robust approach: *Noise-Conditional Score Matching*.

2.1.2.1. Noise-conditional Score Matching

The core idea of noise-conditional score matching is to perturb the data $\mathbf{x}$ with **multiple** levels of Gaussian noise and learn a **single** noise-conditional score model $s_\theta(\mathbf{x}, k)$ to estimate the perturbed score of the density. These multiple scales of noise combine the best of both variants: high noise to get estimates of unknown scores in regions of low density and low noise levels to get precise estimates of scores in regions of high density. The training objective is:

$$\sum_{k=1}^{L} \lambda(k) \mathrm{E}_{p_{\text{data}}(\mathbf{x}), p_{\sigma_k}(\tilde{\mathbf{x}}|\mathbf{x})}[||s_\theta(\tilde{\mathbf{x}}, k) - \nabla_{\tilde{\mathbf{x}}} \log p_{\sigma_k}(\tilde{\mathbf{x}}|\mathbf{x})||_2^2] \tag{2.11}$$

λ is a weighting function, often defined as $\lambda(k) = \sigma_k^2$. After training, we can apply Langevin dynamics, starting with several steps at the highest k noise level with larger step size, before decreasing k and repeating it until we sample a few times at the smallest k with very small step sizes. The resulting model [186] was able to achieve competitive results with GANs on various high-dimensional datasets [185] for the first time and demonstrated the potential of score-based models.

However, there were still several issues with the framework: the number of noise levels is a hyperparameter that is difficult to define and it was not clear what the best schedule is. In addition, the sampling process using Langevin Dynamics requires the same discrete noise levels for training. Instead of choosing these noise levels beforehand, Song et al. [188] proposed the idea to generalize the model towards **infinite** noise levels instead by using continuous time Stochastic Differential Equations (SDEs).

2.1.3. Continuous Time Score-based Diffusion Models

Score-based Diffusion Models (SDMs) are a class of generative model that learn to generate new samples by approximating time reversal of a noising process [188]. They are a generalization of noise-conditional score-matching models, which learn effectively **infinitely** many levels of Gaussian Noise with a **single** noise-conditioned score model $s_\theta(\mathbf{x}, t)$. An SDE describes the diffusion process as a continuous evolution from data p_{data} to noise p_K:

$$dx_t = f(x_t, t)\, dt + g(t)\, dw, \tag{2.12}$$

where $f(x_t, t)$ is the drift term and $g(t)$ is the diffusion term and dw refers to infinitesimal Gaussian noise. At every timestep t, the data sample x_t is deterministically moved by the drift term and randomly changed by the diffusion term.

For every forward SDE there is a corresponding inverse SDE that describes the process going from noise back to the data distribution:

$$d\mathbf{x} = \left[f(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] dt + g(t) d\bar{\mathbf{w}} \tag{2.13}$$

This equation depends on the score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ of the diffusion process, which is learned during training using the denoising score matching objective, discussed before.

Song et al. [188] demonstrated that every SDE has a corresponding Ordinary Differential Equation (ODE), which describes the average probability density at the timestep t:

$$d\mathbf{x} = \left[f(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] dt \tag{2.14}$$

This ODE is the foundation for many fast solvers like DDIM [184], DPM-solvers [115, 116] and follow-up generative models like Consistency Models [187] and is also used in BESO for fast and expressive generation of actions.

There are two major perspectives on Diffusion Models: The score-based perspective that describe the diffusion process using continuous time SDEs, which we covered in detail and the probabilistic view inspired by VAE which is commonly referred to as Denoising Diffusion Probabilistic Model (DDPM) [66, 183] in the literature.

Song et al. [188] first provided a unified perspective on both variants using the SDE formulation. They also showed that both training objectives are conceptually the same except for a different scaling factor. Building on these insights, Karras et al. [83] proposed a principled design for the SDE components. One core insight was to use the noise level as the continuous time variable of the diffusion process $\sigma_t = f(t) = t$. This idea simplifies the denoising objective and results in simple and expressive ODE solvers that achieved SoTA performance.

Further, **E**lucidating the **D**esign Space of Diffusion-Based Generative **M**odels (EDM) introduces improved preconditioning for the neural network. SDMs operate on a large range of noise values from 0.001 to 80. Learning a joint model for this range is unstable given the wide range of predicted output values. This is a consequence of ill-conditioning: at high noise levels around 80, the model must predict small ground truth values, which is challenging while at low noise levels around 0.001, reconstruction is trivial. To address this, EDM introduces input and output preconditioning coefficients:

$$c_{\text{skip}}(\sigma_t) = \sigma_{\text{data}}^2/(\sigma_{\text{data}}^2 + \sigma_t^2) \tag{2.15}$$

$$c_{\text{out}}(\sigma_t) = \sigma_t\sigma_{\text{data}}/\sqrt{\sigma_{\text{data}}^2 + \sigma_t^2} \tag{2.16}$$

$$c_{\text{in}}(\sigma_t) = 1/\sqrt{\sigma_{\text{data}}^2 + \sigma_t^2} \tag{2.17}$$

$$c_{\text{noise}}(\sigma_t) = 0.25\ln(\sigma_t) \tag{2.18}$$

with σ_{data} being 0.5 by default. The full denoising model combines these coefficients with a skip connection and a neural network F_θ:

$$D_\theta(x_t, \sigma_t) = c_{\text{skip}}(\sigma_t) \cdot x_t + c_{\text{out}}(\sigma_t) \cdot F_\theta\big(c_{\text{in}}(\sigma_t) \cdot x_t,\ c_{\text{noise}}(\sigma_t)\big), \tag{2.19}$$

where x_t is the noisy input. The skip connection allows the model to interpolate between directly returning the noisy input (at low noise) and fully

relying on the network prediction (at high noise). The preconditioning ensures that both the input to F_θ and its output target remain at unit variance across all noise levels, stabilizing training. This EDM framework is the direct foundation for BESO (chapter 3), where x_t corresponds to the noisy action $\boldsymbol{a}_t$.

2.1.4. Flow Matching

While continuous-time diffusion models offer a powerful framework for generative modeling, their learned probability-flow ODE trajectories are highly curved [187], making the score field challenging to learn. Flow Matching addresses this by learning straight paths between data and noise [109, 3].

Instead of learning the score of a complex diffusion process, Flow Matching learns a velocity field $v_\theta(z_t, t)$ that transports samples along straight lines from data z_0 to noise $z_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$:

$$z_t = (1-t)z_0 + tz_1, \quad t \in [0, 1]. \tag{2.20}$$

Along these straight paths, the target velocity is constant: $v(z_t) = z_1 - z_0$. The neural network $v_\theta(z_t, t)$ is trained to match this velocity using the conditional flow matching loss:

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{z_0, z_1, t}\left[\|v_\theta(z_t, t) - (z_1 - z_0)\|_2^2\right]. \tag{2.21}$$

At inference, samples are generated by solving the learned ODE $\frac{dz}{dt} = v_\theta(z_t, t)$ with a simple Euler integrator:

$$z_{t-\Delta t} = z_t - \Delta t \cdot v_\theta(z_t, t), \tag{2.22}$$

where $\Delta t = \frac{1}{N}$ for N uniform steps from $t = 1$ (noise) to $t = 0$ (data).

Flow Matching has become the de facto standard for generative modeling in image generation [13], video generation [201], and robotics [11, 161], offering faster convergence and simpler training than score-based diffusion [120]. FLOWER uses Flow Matching for action generation (chapter 6).

2.1.5. Classifier-Free Guidance

Conditional generative models often struggle to strongly adhere to conditioning signals, particularly in high-dimensional settings like image generation where the conditioning information can be diluted across many dimensions. Classifier-Free Guidance (CFG) [67] addresses this by training a single model capable of both conditional and unconditional generation.

During training, the conditioning signal c is randomly replaced with a null token $\varnothing$ with probability $p_{\text{uncond}} \in [0.1, 0.2]$. This dropout scheme teaches the model to generate both $p_\theta(\mathbf{x}|c)$ and $p_\theta(\mathbf{x})$ using the same parameters.

At inference, CFG composes both learned scores together to control the goal-guidance:

$$\tilde{D}_\theta(\mathbf{x}_t, \sigma_t, c) = D_\theta(\mathbf{x}_t, \sigma_t, \varnothing) + w \cdot \big(D_\theta(\mathbf{x}_t, \sigma_t, c) - D_\theta(\mathbf{x}_t, \sigma_t, \varnothing)\big), \quad (2.23)$$

where $w \geq 1$ is the guidance scale. Setting $w = 1$ recovers standard conditional generation, while $w > 1$ amplifies the influence of the conditioning signal at the cost of sample diversity. This formulation requires two forward passes per denoising step which can be batched together for efficiency.

Chapter 3 explores CFG for score-based policies, enabling a single policy to perform both goal-conditioned manipulation and undirected exploration.

2.2. Transformers and Foundation Models

In the prior section we summarized the core principles of DGM with a focus on SDMs. In this section, we cover **Transformers** [198], the neural network architecture that enables model scaling on large datasets. All IL policies used in this thesis are built on a stack of transformer-based components. At the core, all of these models use the same fundamental components: Attention, feed-forward experts, position embeddings and normalization. Further, we summarize the core types of Transformers including MoEs, an important foundation for MoDE. Vision Transformers (ViTs) and CLIP are covered, given their important role in encoding vision and language for policies like MDT in chapter 4.

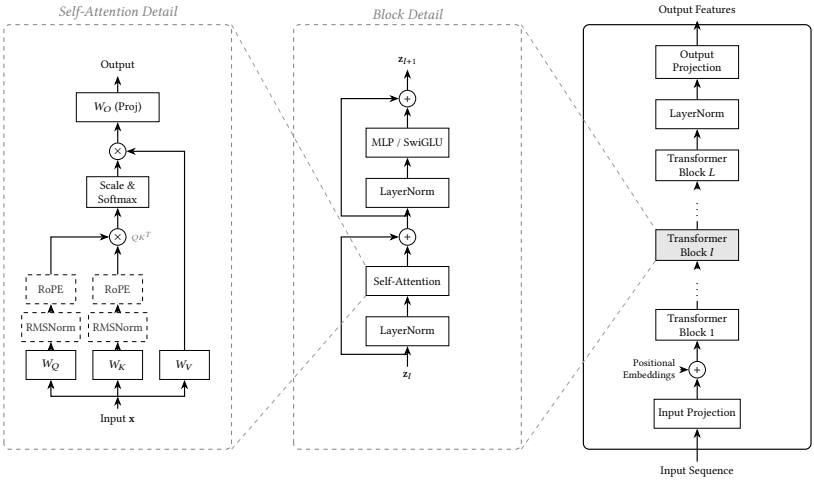


Figure 2.2.: Detailed architecture of the Transformer model. **Right:** the full Transformer consisting of L layers. **Middle:** the internal structure of a single residual Transformer block including normalization and MLP layers. **Left:** The specific operations within the Self-Attention mechanism. The left panel explicitly highlights the projection of Queries, Keys, and Values, along with optional components (dashed lines) such as RMSNorm and RoPE.

2.2.1. Attention Mechanism

The attention part of a transformer layer is the core module for its success in applications like language or action modeling. Figure 2.2 visualizes the full attention computation with optional normalization and position embeddings. Given a set of queries $\mathbf{Q}$, keys $\mathbf{K}$ and value vectors $\mathbf{V}$ from an input sequence $\mathbf{x}$. We can compute the attention score as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{2.24}$$

where d_k is the latent feature dimension of the key vectors. The expressiveness of using a single attention block is limited, thus transformer models use multiple attention heads simultaneously by splitting the latent dimension into

n_h heads. Multi-Head Attention computes several attentions in parallel, which enables tokens to look for different kinds of information more effectively

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O, \tag{2.25}$$

where each head is:

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{2.26}$$

Conceptually, attention can be understood as a differentiable key-value lookup. A query retrieves a weighted combination of values based on a computed similarity metric corresponding their keys.

There are two main types of attention used in almost all transformer models: **Self-Attention** and **Cross-Attention**. Self-Attention refers to the special case where the keys, values and queries are all from the same input sequence of tokens with the objective to update each token with most relevant information from the sequence. In Cross-Attention, the query vectors (**Q**) are generated from one section (e.g. the action tokens of a policy), while the keys (**K**) and values (**V**) are from a different sequence (e.g. image observation tokens of the scene). Thus, the first sequence *queries* relevant information from the second conditioning sequence.

A major drawback of attention and especially, self-attention is its quadratic complexity $O(N^2)$ in both compute and with respect to the sequence length of tokens. This severely limits scalability to longer sequences and has motivated research into more efficient attention mechanisms. Cross-Attention has a complexity of $O(N_q \cdot N_k)$, where N_q refers to the sequence length of query tokens and N_k denotes the number of context tokens, respectively. MDT and FLOWER both employ Cross-Attention for efficient feature extraction from vision and language encoders.

2.2.2. Transformer Experts

Every transformer layer contains Multi-Layer Perceptron (MLP)s to process each individual token after the attention mechanism. In modern Transformers, two types of experts are commonly used: ReLU-MLP and SwiGLU ones. The standard ReLU version is a simple two layered MLP with the equation:

$$\text{FFN}_{\text{ReLU}}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \tag{2.27}$$

It uses the standard ReLU activation function $f_{\text{ReLU}}(\mathbf{x}) = \max(\mathbf{x}, \mathbf{0})$ [137]. The latent dimension of the intermediate layer is commonly defined to be 4 times larger than the input size.

In contrast to the default ReLU experts, the modern SwiGLU [176] one has a slightly different computation that uses two weight matrices, where one is used as gating for the intermediate output of the other one:

$$\text{SwiGLU}(\mathbf{x}) = (\text{SiLU}(\mathbf{x}\mathbf{W}_1) \odot (\mathbf{x}\mathbf{V}_1))\mathbf{W}_2 \tag{2.28}$$

Since we use two weight matrices for the first half of the layer, the intermediate latent multiplier is reduced to 8/3 to account for the increased number of parameters. This dynamic data-dependent gating provides more expressive representations. SwiGLU has become the standard expert variant for many established Large-Language-Models (LLMs) like the Qwen family [155], Kimi K2 [90] or MiniMax [132] as well as diffusion generation models for image generation like Flux [13] or WAN for video generation [201]. MoDE and FLOWER also employ SwiGLU experts for expressive behavior generation.

2.2.3. Transformer Norms

Transformer models consist of many transformer blocks in very deep networks. This results in several training challenges like exploding gradients or vanishing ones. A core part of each transformer layers is layer normalization, which adjusts the value range of latent tokens in each layer.

Given a batched sequence of latent tokens $\mathbf{x} \in \mathbb{R}^{B \times T \times Z}$, where B is the number of samples in a batch, T refers to the number of tokens in the sequence and Z is the latent dimension. In contrast to popular batch norm in Convolutional Neural Network (CNN) [72], which computes the mean and deviation for all samples within a batch, LayerNorm takes the average values of each individual token instead. The initial Transformer proposed the idea of using Layer Normalization:

$$\hat{\mathbf{x}} = \alpha \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{2.29}$$

where α and β are learnable parameters and μ denotes the average value, σ the standard deviation of the data and ϵ refers to a minimum offset of around $1e-6$ for stability near zero [198].

Recently RMS-Norm has gained wide-spread adoption in many Transformer models [219]. RMS-Norm does not use the mean and standard deviation and instead normalizes by the **Root-Mean-Squared (RMS)** of each token. The authors argued that computing the mean and standard deviation in the LayerNorm does not provide performance gains but adds additional computation cost. Instead, the scaling factor α was the most impactful parameter. In RMS-Norm each token vector is scaled by the RMS magnitude:

$$\hat{\mathbf{x}} = \alpha \frac{\mathbf{x}}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} \mathbf{x}_i^2 + \epsilon}} \tag{2.30}$$

By removing the centering (μ) and offset (β) parameters, the normalization is significantly faster to compute. Many Transformers models also include additional RMSNorm in the Attention modules for normalizing the Queries and Keys before the matrix multiplication for increased stability (Figure 2.2 left part). RMS-Norm is used in FLOWER and MoDE to enable stable training on diverse play data

Diffusion Transformer (DiT) architectures often utilize additional **Adaptive Layer Normalization** (AdaLN) [148] to integrate conditioning information. AdaLN replaces the static scale and shift parameters of standard normalization with dynamic values regressed from the condition vector $\mathbf{c}$ (e.g., noise level or class embedding):

$$\text{AdaLN}(\mathbf{h}, \mathbf{c}) = (1 + \boldsymbol{\gamma}(\mathbf{c})) \odot \text{LayerNorm}(\mathbf{h}) + \boldsymbol{\beta}(\mathbf{c}) \tag{2.31}$$

Here, $\boldsymbol{\gamma}(\mathbf{c})$ and $\boldsymbol{\beta}(\mathbf{c})$ are outputs of a lightweight MLP conditioned on $\mathbf{c}$. For improved convergence, the final layer of this MLP is typically initialized with zeros (referred to as **AdaLN-Zero**). This initialization ensures that the transformer blocks initially behave as identity functions, significantly stabilizing the early stages of training. Both FLOWER and MDT employ AdaLN-Zero to condition the policy on the current noise level.

2.2.4. Position Embeddings

Transformers are permutation-invariant and do not have a native understanding of token order. Self-attention treats all tokens in a sequence as equally exchangeable, which is suboptimal in applications such as language

generation or imitation learning, where the position of tokens matters. As a result, Transformers require an additional mechanism to encode position information.

To address this issue, Transformers use different variants of *position embedding*. The original Transformer paper [198] introduced *Absolute Position Embeddings*, which are combinations of sine and cosine functions at different frequencies. For a token at position *pos* and a dimension *i* in the embedding vector, the absolute position embeddings are defined as:

$$PE_{pos,2i} = \sin(pos/10000^{\frac{2i}{d_{\text{model}}}}) \tag{2.32}$$

$$PE_{pos,2i+1} = \cos(pos/10000^{\frac{2i}{d_{\text{model}}}}) \tag{2.33}$$

These sinusoidal embeddings assign a unique, smooth encoding to each position, and the different frequencies across dimensions allow the model to capture both local and long-range positional relationships.

Another type of position embedding used in various transformers is learned position embedding. This type of position encoding is for example used in ViTs. Instead of using predefined formulas like the absolute variant, they create simple embedding matrices of size [Max Sequence Length, Embedding Dim] and let the model learn its optimal vector for each position. The formulation itself is also the bottleneck of learned embeddings, because it cannot extrapolate for longer sequences than its max sequence length. BESO, MDT and MoDE all use learned position embeddings for their action prediction Transformers.

More recently, Rotary Position Embedding (RoPE) [190] has gained widespread adoption in many transformer models, including language [155] and video generation [201]. The core idea of RoPE is to rotate the query and key vectors inside the attention computation instead of adding additional tokens. The degree of rotation is specified by the token's absolute position in a sequence.

For each token's query and key vector (of shape Z), we group its features into pairs of two (e.g. feature 1 & 2, feature 3 & 4, ..). For each feature pair, we apply a 2D rotation matrix:

$$(x'_j + ix'_{j+1}) = (x_j + ix_{j+1}) \cdot e^{i \cdot m\theta_j} \tag{2.34}$$

where $\theta_j = 10000^{\frac{-2j}{d_{\text{model}}}}$.

While RoPE is commonly expressed using complex notation, this rotation is implemented in practice using real-valued operations:

$$\begin{bmatrix} x'_j \\ x'_{j+1} \end{bmatrix} = \begin{bmatrix} \cos(m\theta_j) & -\sin(m\theta_j) \\ \sin(m\theta_j) & \cos(m\theta_j) \end{bmatrix} \begin{bmatrix} x_j \\ x_{j+1} \end{bmatrix} \tag{2.35}$$

This mapping extrapolates well to arbitrary sequence lengths. The attention score is then computed as the dot product of rotated query and keys. It is only dependent on their relative rotation, which is expressed in the difference $\theta_m - \theta_n$ based on their position m and n. Figure 2.2 visualizes the RoPE embeddings in the Attention modules.

RoPE is computationally efficient, since it requires no new learnable parameters and can be implemented as a series of element-wise multiplications. These properties have made RoPEs the de facto default choice as a position embedding for many modern Transformer architectures [155, 201, 90]. Given these properties, FLOWER also employs RoPE for encoding positional information in its Transformer backbone.

2.2.5. Vision Transformer (ViT)

ViTs are a family of Transformers specialized for encoding images. The core idea is to split an image into a grid of fixed-size patches (e.g., 16x16 pixels). These patches are then flattened and linearly projected into the Transformer's latent dimension (e.g., $d = 768$), creating a sequence of "patch tokens" [37].

Because the self-attention mechanism is permutation-invariant, the model needs spatial information. To provide this, 2D position embeddings (such as learned absolute positions or 2D RoPE) are added to each patch token. With these embeddings, the ViT can learn spatial patterns and correlations between all patches of an image.

Many applications, such as image classification, require a single vector to represent the entire image. To accomplish this, a special, learnable `[CLS]` token is prepended to the sequence of patch tokens. During processing, the self-attention mechanism allows this `[CLS]` token to aggregate information from all patches. The final output vector corresponding to this `[CLS]` token is then used as the global summary representation for the entire image.

The core advantage of this approach is achieving architectural homogeneity: by converting an image into a sequence of tokens, the same Transformer architecture used for language can be applied directly to vision. This architectural unification is a key enabler for the unified VLMs and VLAs discussed later in this thesis. In this thesis, ViT-style encoders appear as the vision backbone in MDT and FLOWER.

2.2.6. CLIP: Contrastive Language-Image Pretraining

Contrastive Language-Image Pretraining (CLIP) is a vision-language foundation model that learns a unified latent space of images and text [156] using self-supervised learning. To achieve this, CLIP has been pretrained on a large-scale dataset of around 400M diverse image–text pairs.

CLIP consists of two encoder networks: an image encoder (typically a ViT or CNN) and a text encoder (a Transformer). Each encoder maps its input into an embedding vector, and these embeddings are compared in a shared latent space. Training uses a contrastive InfoNCE loss that encourages matching image–text pairs to have high similarity and non-matching pairs to have low similarity. For a batch of N image–text pairs $\{(\mathbf{I}_i, \mathbf{T}_i)\}_{i=1}^{N}$, the loss is

$$L_{\text{InfoNCE}_i} = -\log \frac{\exp(s(\mathbf{I}_i, \mathbf{T}_i))}{\sum_{j=1}^{N} \exp(s(\mathbf{I}_i, \mathbf{T}_j))} - \log \frac{\exp(s(\mathbf{I}_i, \mathbf{T}_i))}{\sum_{j=1}^{N} \exp(s(\mathbf{T}_i, \mathbf{I}_j))}, \tag{2.36}$$

where $s(\cdot, \cdot)$ denotes the similarity (e.g., cosine similarity) in the shared latent space. All other samples in the batch act as negatives.

Many modern VLM commonly use SigLIP [217], an improved CLIP-style model that replaces the softmax contrastive objective with a sigmoid based one:

$$L_{\text{SigLIP}} = -\frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \left[y_{ij} \log(\sigma(s_{ij})) + (1 - y_{ij}) \log(1 - \sigma(s_{ij})) \right] \tag{2.37}$$

The power of CLIP-style models lies in their generalization. Their embeddings, learned from large-scale image–text pretraining, support tasks such as zero-shot classification and retrieval by comparing similarities in the shared latent space. When used as encoders in robot policies, they enable the policies to interpret novel language commands that are semantically similar to those seen

in pretraining and to select appropriate skills without additional task-specific training.

In this dissertation the CLIP text embedding model is used to encode language goals for BESO, MDT and MoDE. Further, the image model of CLIP is also used to encode image goals for MDT.

2.2.7. Mixture of Experts (MoE)

MoEs are a class of Transformer architectures that modify standard Transformer architectures by replacing dense feed-forward layers with a sparse layer of specialized sub-networks. For each token, the router predicts the top-k experts out of the n available experts to process the tokens (k values of $2-4$ are common in most models). An overview of a MoE-Transformer block is visualized in Figure 2.3. Many widely used LLMs and VLMs use this design including OLMoE [134], Qwen-2.5 [155] and KimiK2 [90].

In a standard top-k MoE layer, the attention and normalization components are identical to those in a dense Transformer. The only structural change is that the single dense MLP is replaced by a router plus a set of smaller expert MLPs. For every token $\mathbf{x}$, the router (a linear layer $\mathbf{W}_r$) computes logits over n experts. These logits are converted to gating probabilities via softmax. The final output $\mathbf{y}$ is a weighted sum of the top-k expert outputs:

$$\text{logits} = \mathbf{x}\mathbf{W}_r \tag{2.38}$$

$$\text{gates} = \text{Softmax}(\text{logits}) \tag{2.39}$$

$$\mathbf{y} = \sum_{i \in \text{TopK}(\text{logits})} \text{gates}_i \cdot E_i(\mathbf{x}) \tag{2.40}$$

MoEs offer several appealing properties for scaling transformers more efficiently: They use significantly fewer FLOPs per forward/backward pass compared to a dense model of the same total parameter size. This also leads to faster training and inference. Further, the MoE design is highly parallelizable because of independent experts, which can be distributed across different GPUs or nodes during training and inference.

The main challenge of using MoEs lies in their training dynamics: jointly learning the router and experts is considerably more difficult than training a dense MLP. Router collapse occurs when the router consistently selects the

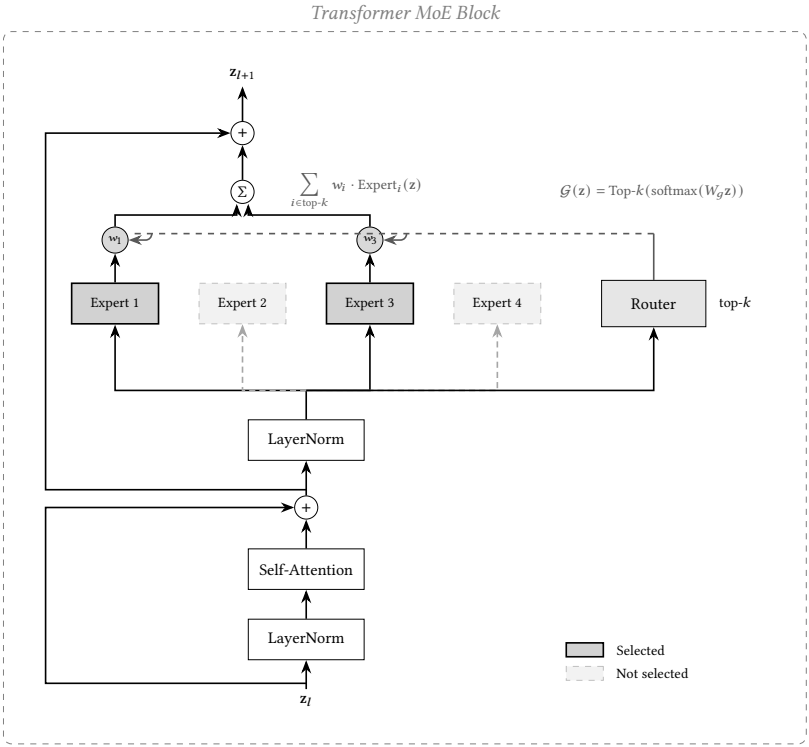


Figure 2.3.: Architecture of a Sparse MoE Transformer block. Following Self-Attention, a learned Router $\mathcal{G}(\mathbf{z})$ activates only the top-k experts (here $k = 2$) via gates w_1, w_3. The weighted sum aggregates outputs from selected experts (solid borders), while inactive experts (dashed) are skipped to scale capacity without increasing computation.

same subset of k experts for all tokens instead of equally distributing them across all available experts. To mitigate this issue, auxiliary load-balancing losses have been introduced. The additional term encourages the model to distribute tokens more equally across all experts:

$$L_{\text{balance}} = \alpha \cdot N \cdot \sum_{i=1}^{N} f_i \cdot P_i, \tag{2.41}$$

where N is the number of experts, f_i is the fraction of tokens assigned to expert i , P_i is the average probability for expert i and α being an additional hyperparameter to control the loss-term weight. The loss is minimized, when probabilities and token fractions are well balanced across all N experts. In large-scale MoE models, each expert typically has a capacity limit, and excess tokens may be dropped or rerouted, which further complicates training and load balancing [90].

In this thesis, we explore novel MoE layers for Diffusion Policy of MoDE (chapter 5) to increase capacity without proportionally increasing inference cost.

2.2.8. Large-Language Models (LLMs)

LLMs are generative models for text generation and understanding. They are AR with Transformer backbones, which predict the probability distribution over all possible tokens. LLM are trained on a large-scale corpus of text using teacher forcing and the cross-entropy loss:

$$L_{CE} = -\sum_{n=1}^{T} \log P(y_n \mid y_{<n}, \mathbf{X}) \tag{2.42}$$

Teacher forcing uses a ground-truth input sequence for next-token prediction. The LLMs commonly uses a decoder-only transformer model using causal attention. Causal attention refers to a variant of self-attention, where any token can only attend to itself and tokens earlier on the sequence. This guarantees that the next word prediction is only dependent on the prior sequence of words.

For tokenization of text, Byte-Pair Encoding (BPE) is commonly used. Given a large text corpus, BPE starts from a base vocabulary (often characters) and iteratively merges the most frequent adjacent token pairs. These frequent character or subword pairs are merged into new tokens, and the final set of tokens forms a discrete vocabulary. For example, the word *robotics* might be tokenized into `[robot]` and `[ics]`, where `[robot]` is a frequent subword and `[ics]` is a common suffix. This method strikes a balance between having a compact vocabulary and preserving semantically meaningful units.

Modern LLMs use RoPE position embeddings, which are covered in Section 2.2.4. LLMs pretrained on internet-scale text data encode generalist

world knowledge and are capable of reasoning [141] and understanding general patterns [133] providing huge potential for versatile robot agents. In this thesis, we leverage LLM indirectly, through VLMs and VLAs that use LLM decoders for feature processing of images and text.

2.2.9. Vision-Language Model (VLMs)

VLMs are a class of models for joint vision and text understanding. VLM combine pretrained LLM with pretrained vision encoding models like CLIP [2, 112].

To translate vision tokens to the latent space of the LLM, VLMs use projector networks. The most common projector used in VLMs is a simple MLP. The projector acts as a vision to language translator, which projects the ViT tokens to the LLM space. The processed vision tokens are then concatenated with the language token sequence. For clear distinction between both modalities, special tokens that symbolize beginning of an image `[BOI]` and end of an image `[EOI]` are introduced. The multimodal token sequence is then processed by the LLM layers, enabling LLMs to learn joint understanding of vision and language.

VLMs are commonly trained in several stages. In the first stage, both ViT and LLM are kept frozen and only the projector is trained. This stage teaches the LLM to process images. Next, in Stage 2, the ViT and LLM layers are unfrozen and trained jointly. This second stage training is done using high-quality instruction following data. It enables the VLM to talk and reason about images or solve different tasks like bounding box prediction or pointing [33].

VLMs hold great potential for many robotics tasks given their unified vision-and-language understanding. They are used as task-planners [39], for labeling trajectories [14] and as universal encoders for manipulation policies including FLOWER and other VLAs [11, 89, 10], which will be discussed in the next section in detail.

2.2.10. Vision-Language-Action Models (VLAs)

VLAs represent a novel class of foundation models for robot manipulation. The definition of VLAs varies in the literature. In this thesis, we propose the following definition:

A VLA is a neural network utilizing a backbone jointly pretrained on large-scale vision-language data, which is subsequently fine-tuned to generate control commands. These commands may range from low-level robot joint positions to high-level actions for virtual agents.

This definition implicitly includes two general groups of VLAs: the standard design using a pretrained VLM backbone and Video-Action Policies [24, 69] that leverage pretrained video-generation backbone. Crucially, we distinguish these from approaches that utilize separately pretrained encoders (e.g., combining CLIP-text with DINOv2 image embeddings). This definition strongly contrasts to models that lack joint vision-language pretraining. We classify the latter as *multimodal policies* rather than VLAs for the remainder of this work (e.g., MDT and MoDE are multimodal policies but not VLAs).

Most VLAs belong to two general types: discrete-tokens ones and flow/diffusion-based ones [11, 10, 161]. Discrete tokens variant discretizes actions using either binning [18, 89], k-means clustering [172, 31] or VQ-VAEs [98]. In these VLAs, action tokens are added to the existing vocabulary of the VLM, and the policy does not require additional modules for action generation. This auto-regressive approach can represent action generating in the native next-token prediction formulation of LLMs and VLMs, which results in faster convergence [151]. The core disadvantage is discretizing longer action chunks often results in less expressive behavior and slow inference because of the iterative next-token prediction setup.

Flow/Diffusion-based VLAs use additional parameters for action generation, which is conditioned on the latent features of the VLM. While training additional parameters complicate training dynamics, it offers the core advantage of fast and expressive generation of longer action chunks. FLOWER is a novel, efficient VLA that belongs to the flow/diffusion category and also addresses the issue of slower convergence using intermediate fusion (chapter 6).

2.3. Imitation Learning

IL summarizes all methods that learn to imitate recorded behavior using neural networks. We have three core categories of algorithms: Behavior Cloning, which will be discussed in detail next subsection, Inverse Reinforcement Learning (IRL) [5] and Generative Adversarial Imitation Learning (GAIL) [65]. Both GAIL and IRL require environment interactions which makes them impractical for learning more complicated real-world behaviors [31]. This thesis focuses exclusively on Behavior Cloning (BC), given its offline paradigm to learn from static datasets without additional real-world rollouts. Below we will summarize the core algorithm, its challenges and most relevant benchmarks.

2.3.1. Behavior Cloning (BC)

BC defines a class of machine learning methods that aim to replicate the seen demonstrations [153, 142]. Formally, BC learns from a dataset $D = \{\tau_i\}_{i=1}^{K}$ of K expert trajectories, consisting of a set of state-and-action pairs $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i)\}_{i=1}^{N}$. There are two major losses used in BC training:

- **Continuous Action Prediction:** The most common approach is to learn a policy $\pi_\theta(s_i)$ to minimize the Mean Squared Error (MSE) to the expert action a_i:

$$L_{BC} = \frac{1}{N} \sum_{i}^{N} ||\pi_\theta(s_i) - a_i||_2^2 \tag{2.43}$$

  While this formulation is expressive, it fails to learn multimodal solutions [78]. The MSE objective produces mode-averaging behavior. Given a simple example of moving around a tree with an action distribution of going 50% around the left and 50% going around the right side, the policy will average both modes and learn to go straight.

- **Discrete Action Prediction:** For discrete actions we use the Cross-Entropy Classification loss, identical to the LLM setup for language prediction:

$$L_{BC-NLL} = -\frac{1}{N} \sum_{i=1}^{N} \log(\pi(a_i|s_i)) \tag{2.44}$$

The cross-entropy loss does not produce mode-averaging behavior and can maintain multimodal solutions effectively. However, these approaches trade multimodality with expressiveness. Discrete tokens require compression compared to continuous values, and the learned behavior is limited to pre-defined bins, which can result in low precision for more fine-grained behavior.

The core challenges of BC are [142]:

- **Multimodal Behavior:** First, BC struggles with the multimodality of generated actions. Many tasks can be solved using different strategies and movements, and an ideal policy needs to imitate all possible solutions. Traditional log-likelihood optimization produces mode-averaging behavior, which cannot learn all possible solutions and learns an average of all observed modes. This can result in low success rates and wrong and potentially harmful behavior.
- **Covariate Shift:** The shift between seen states and observed states during rollouts is another core issue of BC. The policy is not able to understand what to do from these unknown states and will fail to complete the task.
- **Compounding Errors:** During rollouts the learned policy needs to continuously generate new actions. These generated actions will always be slightly off compared to the ground truth. These small prediction errors accumulate over time and drift the policy out of seen states or cause failures in behavior.

These challenges are also motivations for the contributions in this thesis: Chapter 3 tackles the issue of multimodal policies using expressive score-based diffusion policies. Our image-based policies use action chunking [27] to counter compounding errors by predicting a sequence of actions at every prediction step; the state-based BESO experiments use a single-step action horizon ($H = 1$). Further, MoDE and FLOWER try to tackle the covariate shift issue in chapter 5, and chapter 6 by pretraining on large-scale diverse data to make the policy more robust.

2.3.2. Goal-Conditioned Imitation Learning (GCIL)

The objective of GCIL is to derive a general-purpose goal-conditioned policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$ from a dataset of goal-labeled state-action triples $\mathcal{D} = [\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{g}]_{i=1}^{N}$. This is the general problem formulation used throughout this thesis. We aim to learn a goal-conditioned policy $\pi_\theta(\boldsymbol{a}_{i:i+H-1} \mid \boldsymbol{s}_{i-h+1:i}, \boldsymbol{g})$ to predict a sequence of actions with horizon H conditioned on a history of h past observations $\boldsymbol{s}_{i-h+1:i}$ and a goal $\boldsymbol{g}$. The goal $\boldsymbol{g}$ could be either an image of the goal-state $\boldsymbol{o}$, a future state vector $\boldsymbol{s}^*$, or an encoded language instruction $\boldsymbol{l}$.

The action horizon H and history length h are core design choices that vary across chapters. For state-based environments in BESO and MoDE, we use reactive policies ($H = 1$) with history conditioning ($h \in [2, 10]$) to capture temporal dependencies from proprioceptive observations. For image-conditioned manipulation, we employ action chunking ($H \in [10, 50]$) with stateless policies ($h = 1$), where the chunk length depends on the data collection frequency. The stateless setup is optimal for image-based GCIL as single frames typically contain sufficient information for action prediction. We specify the choice of H and h in every chapter and omit the subscripts for simplicity.

The general training objective is to maximize the log-likelihood of the expert's action chunk given the observation and goal:

$$\max_\theta \mathbb{E}_{(\boldsymbol{a},\boldsymbol{s},\boldsymbol{g})\in\mathcal{D}} \left[\log \pi_\theta(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})\right] \tag{2.45}$$

2.3.3. Learning from Play (LfP)

LfP is a special case of GCIL with diverse data covering many skills but not many segmented and labeled trajectories. Instead the dataset consists of long sequences with many skills performed in random orders.

To learn GCIL behavior from this uncurated, task-agnostic data [119], we sample future images as desired goals for the policy. We start with a long, unlabeled trajectory $\boldsymbol{\tau} = \{\boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_I, \boldsymbol{a}_I\}$. To create a goal-labeled state–action triplet $(\boldsymbol{s}_{i-h+1:i}, \boldsymbol{a}_{i:i+H-1}, \boldsymbol{s}_j)$, we first sample a random index i that determines the current state $\boldsymbol{s}_i$ and action sequence $\boldsymbol{a}_{i:i+H-1}$. Next, we sample a goal index j such that $j > i$, and define the goal as the future state $\boldsymbol{g}_i = \boldsymbol{s}_j$. This

teaches the policy to reach any desired future state. For sampling these future states two strategies are commonly employed:

- **Random Sampling:** The desired goal-state j is sampled uniformly from $[i+1, I]$. This teaches the policy to solve both near and far goals.
- **Geometric Sampling:** We restrict the future horizon for sampling and use a geometric distribution that assigns higher probability to nearer future states within this range. This emphasizes shorter temporal offsets while still sampling more distant goals. It is the technique used for MDT goal sampling.

This LfP framework provides a dense source of self-supervised goal-conditioning. In this thesis, we explore how this dense signal can be combined with the sparse, manually-annotated language labels $\boldsymbol{l}$ that are also available in datasets like CALVIN to learn policies in chapter 4. To address the temporal consistency issues of single-step policies $\pi(\boldsymbol{a}_i \mid \boldsymbol{s}_i)$, our image-based policies use **Action Chunking** [222, 27]. The policy predicts H future actions $\boldsymbol{a}_{i:i+H-1} \in \mathbb{R}^{H \times d}$ from the current observation $\boldsymbol{s}_i$, where d is the action dimension. The horizon depends on the benchmark and policy rather than a fixed physical duration: MDT predicts $H = 10$ actions in CALVIN, whose control rate is 30 Hz [130, 163], and uses $H = 20$ for its real-robot experiments at 6 Hz. The experiment-specific horizons and execution settings are given in the corresponding chapters.

2.3.4. Action representations

The choice of action representation for IL has high relevance for the performance. In this section we briefly summarize the two most relevant action spaces for imitation learning that are also used in this dissertation.

2.3.4.1. Joint Space Representation

In this action space, policies directly predict the target joint angles of the robot manipulator $\mathbf{a} \in \mathbb{R}^{d_{dof}}$. The core advantage is simplicity: it avoids the need for Inverse Kinematics (IK) computations, which can be computationally expensive or unstable near singularities. However, it is less generalizable across different robot morphologies. The ALOHA benchmark used in chapter 6 utilizes joint-space actions to enable precise bi-manual manipulation at

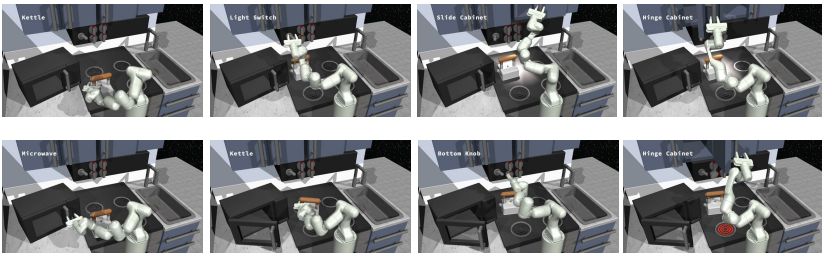


Figure 2.4.: Relay Kitchen example rollouts. Two example rollouts in the Relay Kitchen benchmark introduced by Gupta et al. [57], illustrating different task compositions.

high frequencies (50Hz) [222]. In addition our real robot setup summarized in Section 2.3.5.6 also uses joint state representation.

2.3.4.2. Delta-End-Effector Representation

This representation predicts the relative change in the robot's end-effector pose $\Delta\mathbf{p} \in \mathbb{R}^6$. It consists of a translation vector $\Delta\mathbf{x} \in \mathbb{R}^3$ and a rotation component $\Delta\mathbf{r} \in \mathbb{R}^3$. While rotations can be represented via Quaternions, most benchmarks in this thesis (e.g., CALVIN [130] and LIBERO [110]) and the majority of the OXE dataset [30] utilize Euler Angles. Although Euler angles are not continuous (suffering from singularities and wrap-around issues globally), they are stable and data-efficient for predicting small, relative rotation changes common in pick-and-place tasks.

2.3.4.3. Gripper Representation

Most robots represent the gripper as a binary signal attached to the action vector as an additional dimension. It typically takes discrete values $g \in \{0, 1\}$ (open/close) or continuous values $g \in [-1, 1]$ representing the width.

2.3.5. Simulation Benchmarks

In this section we provide an overview of the most frequently used imitation learning benchmarks in this dissertation to evaluate different aspects of policy learning. We select benchmarks that test different aspects of policy learning: Block Push and Relay-Kitchen for multimodality, CALVIN for long-horizon language conditioning and generalization, LIBERO for multi-task learning, and SIMPLER for zero-shot evaluation of pretraining of foundation policies.

2.3.5.1. Relay Kitchen

Relay Kitchen [57] is a multi-task simulation benchmark using a Franka Emika Panda robot in a virtual kitchen environment. The robot manipulates seven objects: kettle, microwave, sliding door, hinged door, light switch, and two burners. The dataset consists of 566 human-collected demonstrations where each trajectory contains a sequence of four manipulation tasks.

The robot is controlled through a 9-dimensional action space (joint and end-effector positions), and observations are 30-dimensional state vectors encoding object configurations. Policies are conditioned on randomly sampled future states as goal states. Evaluation uses two metrics: *result* measures how many desired goals are achieved per rollout (maximum 4 credits), while *reward* measures overall performance by crediting any valid goal completion in the environment (in theory maximum 5 credits, but all recorded demonstrations only contain sequences of 4).

2.3.5.2. Block Push

Block Push [47] is a PyBullet environment where an XArm robot must push two colored blocks (red and green) into correspondingly colored square target areas. The dataset contains 1,000 demonstrations collected by a deterministic scripted controller that randomly selects which block to push first and which target to use, resulting in four distinct behavioral modes plus additional stochasticity from varied pushing trajectories.

Policies receive state-based observations and are conditioned on goal states specifying the desired block-target configuration and pushing order. A block is considered successfully placed if its center lies within 0.05 units of the target

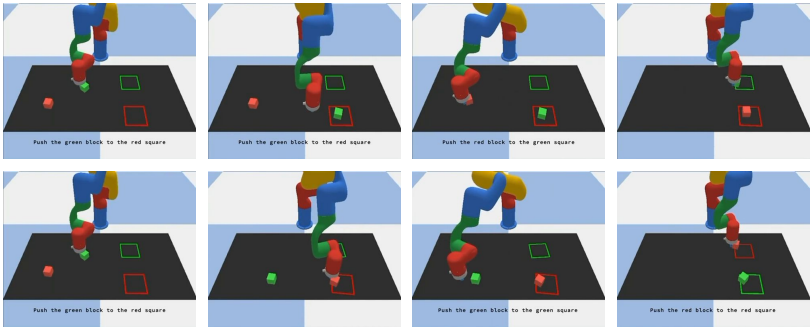


Figure 2.5.: Block Push example rollouts. Two rollouts in the multimodal Block Push benchmark based on Implicit Behavioral Cloning [47, 172], showing different block-to-target assignments and execution orders.

square. Evaluation credits 0.5 points per correctly placed block (maximum 1.0), requiring both correct target placement and execution order.

2.3.5.3. CALVIN Benchmark



(a) Environments **(b)** Example CALVIN-Rollout

Figure 2.6.: Overview of the CALVIN environment. (a) CALVIN contains four different settings (A,B,C,D) with different configurations of slides, drawers and textures. (b) Example rollout consisting of 5 tasks in sequence. The next goal is only given to the policy if it completes the prior goal. The environment images and rollout illustrate the CALVIN benchmark introduced by Mees et al. [130].

CALVIN [130] is a GCIL benchmark for learning language-conditioned manipulation from human play data. Unlike structured demonstration datasets, CALVIN consists of unstructured interaction trajectories where humans freely explore four distinct environments (A, B, C, D), each with unique configura-

tions of drawers, buttons, and sliders (Figure 2.6). The dataset defines 34 basic skills (e.g., "rotate blue block right," "open drawer," "turn off light bulb") across very long-horizon trajectories, that consists of play interactions that last up to 1 hour. For language-conditioned learning 1% of the data is segmented into equal long short trajectories of 64 timesteps and provided a task description in natural language.

Evaluation protocol. Policies are evaluated on sequences of 5 consecutive tasks sampled from the 34 skills. The environment reveals the next task only after successful completion of the current one. Evaluation runs 1000 rollouts in environment D and reports both the full-sequence success rate and the average number of completed tasks per sequence. Policies observe the environment state via static and wrist-mounted camera images.

Dataset splits. We use three standard splits throughout the thesis for testing different aspects of policy learning:

D→D: 5,124 labeled windows from environment D only. The small scale makes learning and composing all 34 tasks challenging.

ABC→D: 17,842 sequences from environments A, B, C with zero-shot evaluation in the unseen environment D. Tests generalization to novel scene configurations and is the most challenging setup in CALVIN.

ABCD→D: 22,966 sequences from all four environments. Tests scaling behavior with maximal data diversity.

Given the fair standardized evaluation protocol, baseline numbers are directly reported from respective papers unless stated otherwise.

2.3.5.4. LIBERO Benchmark

The LIBERO task suites [110] consists of 5 different ones in the benchmark with 50 demonstrations per task. For chapter 4, that uses both language goals or image goals, we generate goal images by utilizing the final state of each rollout. We used the default end-effector action space in all our experiments. The benchmark, visualized in Figure 2.7, is structured into five distinct task suites, each designed to test different aspects of robotic learning and manipulation:



Figure 2.7.: Overview of the LIBERO benchmark. Representative tasks from the Goal, Object, Spatial, and LIBERO-100 suites. Images adapted from Figures 8–11 of Liu et al. [110] (cropped, rearranged, and rescaled). The LIBERO-100 montage illustrates tasks used in the 90/Long split.

- **Spatial**: This suite emphasizes the robot's ability to understand and manipulate spatial relationships. Each task involves placing a bowl, among a constant set of objects, on a plate. The challenge lies in distinguishing between two identical bowls that differ only in their spatial placement relative to other objects.
- **Goal**: The Goal suite tests the robot's understanding and executing varied task goals. Despite using the same set of objects with fixed spatial relationships, each task in this suite differs in the ultimate goal, demanding that the robot continually adapt its motions and behaviors to meet these varying objectives.
- **Object**: Focused on object recognition and manipulation, this suite requires the robot to pick and place a unique object in each task.
- **Long**: This suite comprises tasks that necessitate long-horizon planning and execution. The Long suite is the most challenging variant of LIBERO, as it tests the robot's ability to maintain performance and adaptability over extended task duration that require action generation for 450 steps.
- **90**: Offering a diverse set of 90 short-horizon tasks across five varied settings, that require the policy to acquire diverse skills using a single policy.

Given the standardized evaluation protocol, we directly compare our results against baselines values reported in their respective papers.



Figure 2.8.: SIMPLER task overview. Simulation frames adapted from the supplementary videos accompanying Li et al. [106]. The left four images show the WidowX/BridgeData V2 tasks; the right four show Google Robot tasks.

2.3.5.5. SIMPLER Benchmark Tasks

The real2sim benchmark SIMPLER [106] consists of two evaluation settings.

Google Robot Setting. This setting comprises four manipulation tasks. *Pick coke can* requires grasping and lifting an empty Coke can from the table across 75 trials, testing three orientations (horizontal, vertical, standing) at 25 grid points each. *Move objects near objects* evaluates relative positioning over 60 trials with three objects in triangle configurations (upright and inverted). Eight objects are used (blue bottle, Pepsi can, orange, 7up can, apple, sponge, Coke can, Redbull can) in five predefined triplets. *Drawer manipulation* tests articulated object handling over 54 trials: the robot is positioned at nine floor locations and must open or close a specific drawer (top, middle, or bottom). *Drawer + placement* combines sequential actions over 27 trials: open the top drawer, then transfer an apple from the cabinet surface into the drawer, with the robot at three locations and the apple at nine grid points.

For this setting, SIMPLER defines two evaluation variants to mitigate the real-to-sim gap. In the *Visual Matching* variant, simulated observations are composited with real-world backgrounds and baked object textures to closely replicate the appearance and lighting of real camera images, thereby reducing visual domain shift. The alternative *Variant Aggregation* variant evaluates policies across a diverse set of randomized simulation variants—varying camera poses, lighting, backgrounds, and object textures—and aggregates

results over all runs. While both yield meaningful correlations to real-world success, Li et al. [106] report that Visual Matching achieves higher ranking fidelity (lower mean maximum rank violation and higher Pearson r) than Variant Aggregation on the Google Robot tasks.

WidowX + Bridge Setting. This setting features four tasks over 24 trials each. *Put spoon on towel* requires placing a spoon from one corner to another of a 15cm square, with initial orientation alternating between horizontal and vertical. *Put carrot on plate* follows the same structure but substitutes a carrot and plate. *Stack blocks* tests precise stacking of a 3cm green block on a yellow block, using two square configurations (10cm and 20cm side lengths) with blocks at different corners. *Put eggplant into yellow basket* places an eggplant randomly in the right sink basin and a basket in the left basin, with varying eggplant locations and orientations kept graspable.

2.3.5.6. Real World Play Kitchen Setup



Figure 2.9.: Overview of the real robot play kitchen setup. The left image shows the play kitchen with its relevant objects in the scene, while the right image shows the cameras and the second robot used for teleoperation. The policies are conditioned on the side-view camera and a birds-eye view camera.

Detailed Environment Overview. Our real robot setup with the toy kitchen is visualized in Figure 2.9. In the kitchen the robot can interact with the microwave positioned on the top right, the oven in the lower left half of the kitchen, the cooler on the lower-right side of the kitchen and the sink on the right side of the counter top. The robot is positioned next to a toy kitchen with the following additional objects: a banana, a pot, a toaster with toast. For the MDT experiments, we create a set of 20 diverse tasks for the robot to

learn from the partially labeled play data. All tasks of our dataset are shown in Figure 2.10.

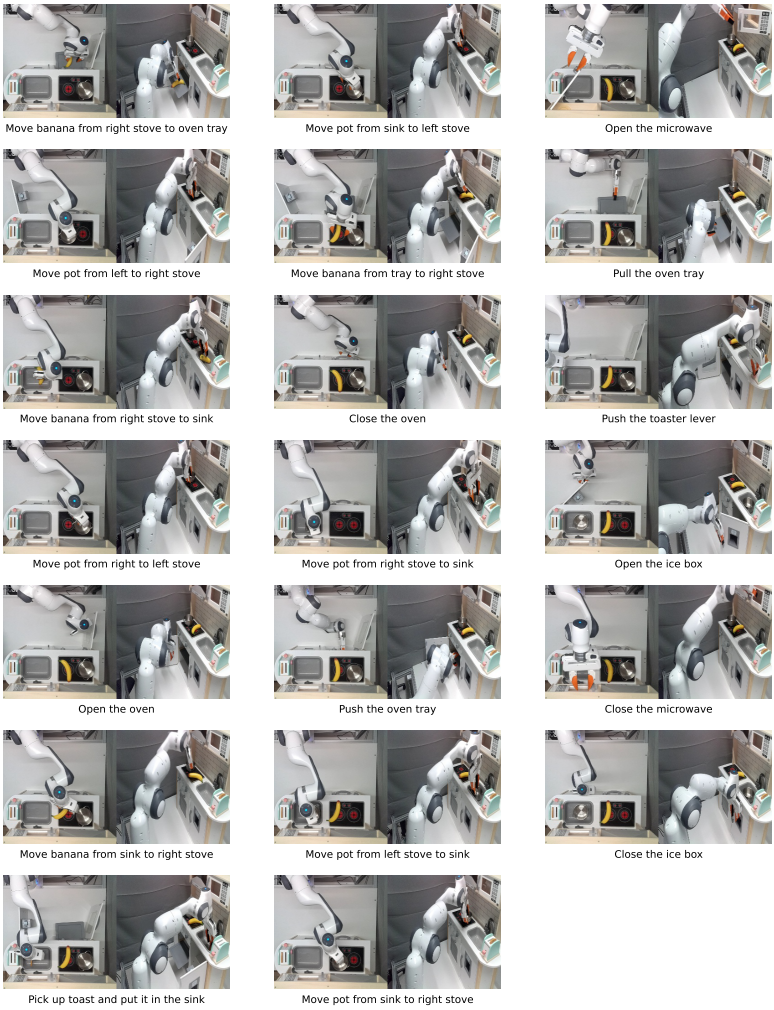


Figure 2.10.: Overview of the 20 tasks recorded in our real world kitchen. Both camera perspectives used for training policies are recorded.

Play Dataset Collection. Our volunteers collect play data with teleoperation with a leader and follower robot setup, which is visualized in Figure 2.9. During the teleoperation, we collect the robot's proprioceptive sensor data and two images from our two static cameras with 6 Hz. We extract the desired joint position as our action signal and normalize it in the range $[-1, 1]$. To label the data with additional text instructions, we randomly sample short sequences from our uncurated play dataset and ask a human to describe the task in this segment. For every task we query GPT-4 to generate different text instructions for more diverse language descriptions. We note, that training a single policy on such a small dataset of real world play data is very challenging for the models.

3. Goal-conditioned Imitation Learning using Score-based Diffusion Policies

The foundation of a generalist manipulation agent lies in a robust policy representation capable of efficiently learning from diverse, unstructured play data (**C1**).

At the time of this work, existing imitation learning policies struggled to model multimodal behavior from uncurated play data. This chapter addresses the critical question of designing an effective policy backbone for robot learning by introducing **BESO** (**BE**havior generation using **S**core-based diffusi**O**n models), a novel policy representation that leverages score-based diffusion models to directly model complex action distributions. Conducted concurrently with the earliest diffusion-based policy work [27, 147], this chapter presents the first score-based diffusion models for goal-conditioned imitation learning. Our results highlight the importance of expressive, multimodal policy representations for learning from play. BESO serves as a scalable and data-efficient foundation upon which subsequent chapters build with multimodal conditioning (chapter 4) and efficiency improvements (chapter 5).

*The content of this chapter is based on the paper **Goal-Conditioned Imitation Learning using Score-based Diffusion Policies** (Moritz Reuss, Maximilian Li, Xiaogang Jia, Rudolf Lioutikov, 2023), published in **Robotics: Science and Systems (RSS)**. The paper received the Best Paper Award at the ICRA 2023 Workshop on Learning from Diverse, Offline Data. Wording, notation, and formulations were revised for this thesis.*

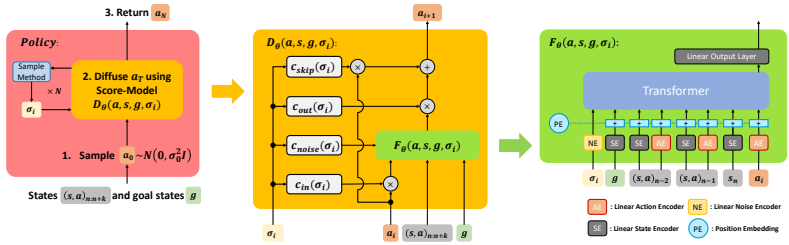


Figure 3.1.: Overview of the action generation process of BESO with the used model architecture. Left: General Action Generation Process using the Diffusion Process to denoise the next action given the current observation $\boldsymbol{s}$ and desired goal state $\boldsymbol{g}$ in N-steps. **Middle:** the high-level score model with its pre-conditioning layers and skip connections. **Right:** the internal denoising score model, which uses a transformer architecture with causal masking to iteratively predict the denoised action given the sequence of prior observations, actions, and the goal sequence.

3.1. Introduction

Prior policy representations for learning from play data face a fundamental trade-off: expressive models like VAEs [57, 130, 150] or discrete tokenizers [31, 172] require supplementary networks or hierarchical structures to capture multimodal behavior. This results in multiple learning objectives and additional tuning challenges.

This chapter introduces **BE**havior Generation using **Sc**o**O**re-based Diffusion models (BESO), a score-based diffusion policy that addresses this limitation. By modeling the action distribution directly with score-based diffusion models, BESO captures multimodal behavior in a single network without hierarchical decomposition. BESO uses SDMs [183, 66, 188, 83], that progressively diffuse data to noise through a forward SDE. By training a neural network, known as the score or denoising model, to approximate the score function, one can reverse the SDE to generate new samples from noise in an iterative sampling process. Modeling the goal-conditioned action distribution using a score-based diffusion model has several core benefits for policy learning.

First, the expressiveness and multimodality of SDMs are crucial for effective conditioned behavior generation. On several challenging goal-conditioned

benchmarks, including the conditioned Relay Kitchen and CALVIN benchmark [130], BESO consistently outperforms SoTA methods across 54 tasks in 5 different benchmarks in simulation and real-world environments.

Second, by leveraging Classifier-Free Guidance Training of SDMs, BESO effectively learns two policies simultaneously: a **goal-dependent** policy and a **goal-independent** policy, which can both be composed together or used independently.

Third, our model is easy and stable to train with a single training objective without additional rewards. This contrasts with other generative models, such as Implicit Behavior Cloning (IBC) [47], or hierarchical policies [57].

Fourth, SDMs do not restrict the choice of the model architecture as in other generative models such as VAEs or EBMs [47]. BESO leverages two novel Transformer architectures augmented with preconditioning to predict flexible-length action chunks.

Finally, BESO can diffuse new actions fast. While concurrent diffusion-based policies [147] require 30+ denoising steps per action, BESO outperforms SoTA policies using only 3 denoising steps.

To summarize our contributions:

- BESO, a new policy representation based on score-based diffusion models for effective goal-conditioned behavior generation from uncurated play data.
- Use of Classifier-Free Guidance-based Diffusion Policy to simultaneously learn a goal-dependent and goal-independent policy from play.
- Systematic evaluation of key components for fast and efficient action generation using Score-based Diffusion policies combined with extensive experiments and ablations.

Code, pre-trained models, and example rollouts are available at `https://intuitive-robots.github.io/beso-website`.

3.2. Related Work

GCIL. It is a sub-domain of imitation learning [142, 4] where demonstrations are augmented with goal states that indicate the intended task. These goal states disambiguate demonstrations that might otherwise appear similar, enabling a single goal-conditioned policy to exhibit different behaviors depending on the specified goal. A related paradigm is Goal-Conditioned Reinforcement Learning (GCRL) [45, 44, 150, 86], which uses reward signals rather than demonstrations. However, reward signals are difficult to define for complex manipulation tasks, and the required policy rollouts are expensive in real-world settings—making demonstration-based approaches more practical for our target domain.

Similarly, several works explore Goal-Conditioned Policy Learning in an Adversarial Imitation Learning Setting. GoalGAIL [35] uses a GANs framework to match goal-conditioned occupancy measurements, learning a unified policy for various goals. Lee et al. [99] proposes an improved goal-proximity measurement to inform agents about task progress. Go-Far [122] formulates GCIL as a state-occupancy matching problem between a dynamics-constrained agent and an idealized expert. More recently, GO-DICE [73] proposed an option-aware framework that also considers sub-goals for effective goal-conditioned learning. These algorithms are all limited to simple state-based environments, while BESO can scale effectively to large-scale image benchmarks in simulation and the real world to learn a diverse range of language-conditioned tasks.

Learning from Play. The goal of LfP [119] is to learn goal-specified behavior from a diverse set of unlabeled state-action trajectories. Play data is characterized by unsegmented, multimodal trajectories. This makes learning meaningful behaviors more challenging, as the policies need the ability to deal with multiple ways of solving a task, distinguish between similar ways to solve different tasks, as well as the ability of long-horizon planning to reach goals far into the future. Prior work aimed to extract representations from play data for effective downstream policy learning [215] or learned self-supervised representations of skills, referred to as latent plans, using Conditional Variational Autoencoders (CVAE) [57, 119, 130, 129]. Transformer-based architectures were also researched as a policy class for task-agnostic behavior learning [31, 172]. Another body of work tries to improve LfP, by focusing on the data

aspect and learning from object-centric interactions, instead of randomly sampled sequences [8].

Generative Models in Policy Learning. Imitation Learning can be formulated as a state-occupancy matching problem, where the goal is to learn a policy that matches the state-occupancy distribution of expert demonstrations. The unknown expert demonstration can now be approximated through modern generative model architectures. One popular approach is the use of GAN [65, 48]. These methods use a generator policy that learns to imitate the observed behavior of the expert and a discriminator, which distinguishes between real and fake trajectories. They require extensive rollouts during training, which is not feasible in our setting. Other approaches use CVAEs [125, 150, 57, 129, 167] to learn a latent embedding to represent the underlying skills. Recent work also applied Energy-based models as conditional policies for behavior cloning [47]. Normalizing flows have also been proposed as a policy representation [182].

Diffusion Generative Models in Robotics. Most approaches that apply diffusion models in robotics applications focus on the discrete DDPM variant [66]. The DDPM Diffusion model has been used in Offline-RL to generate state-action or state-only trajectories using large U-net architectures [76, 1]. DDPM has also been applied as a policy regularization method in a step-based Offline-RL setting in combination with a learned Q-function [203]. Recently, score-based generative models have been leveraged to synthesize cost functions for grasp pose configurations [195]. In addition, Conditional score-based generative models have been proposed to learn the reward function for inverse reinforcement learning [87].

The closest concurrent work to BESO is Diffusion Policy [27] and Diffusion-BC [147], which both propose the use of conditional, discrete DDPM as a new policy class for Behavior Cloning. However, both of these methods have slow action generation: Diffusion-BC synthesizes new actions in 50 stochastic sampling steps and Diffusion Policy recommends up to 100 denoising steps. BESO leverages the *probability flow* ODE combined with fast, deterministic samplers and optimized noise levels. Hence, BESO requires significantly fewer function evaluations in every action prediction.

3.3. Method

In this section, we describe our approach to goal-conditioned behavior generation using Score-based diffusion models.

3.3.1. Score-based Diffusion Policies

Our goal is to learn the policy distribution $\pi_{\mathcal{D}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{g})$ underlying the play dataset $\mathcal{D}$. We achieve this by defining a continuous diffusion process that gradually adds Gaussian noise to actions sampled from $\pi_{\mathcal{D}}$, producing a sequence of intermediate distributions p_t for $t \in [0,T]$. At $t=0$, we have $p_0 = \pi_{\mathcal{D}}$ (the data distribution); at $t=T$, we have $p_T \approx \mathcal{N}(\mathbf{0}, \sigma_T^2\mathbf{I})$ (an unstructured Gaussian prior).

Following Song et al. [188], this diffusion process can be described as a SDE. We adopt the formulation of Karras et al. [83]:

$$d\boldsymbol{a} = \big(\beta_t\sigma_t - \dot{\sigma}_t\big)\sigma_t \nabla_a \log p_t(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{g})dt + \sqrt{2\beta_t}\sigma_t d\omega_t, \tag{3.1}$$

where $\nabla_{\boldsymbol{a}} \log p_t(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{g})$ refers to the score-function, ω_t is the Standard Wiener process, which can be understood as infinitesimal Gaussian noise. The noise scheduler is denoted by σ_t, and $\beta(t)$ describes the relative rate at which the current noise is replaced by new noise. In our approach, we set $\sigma_t = t$, following prior work in image generation [83]. Here, $\dot{\sigma}_t = \frac{d\sigma_t}{dt}$ denotes the time derivative of the noise schedule.

At every timestep t and related noise level there exists a corresponding marginal distribution $p_t(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{g})$, which is the result of injecting Gaussian noise to samples from p_{play}. This can be expressed as $p_t(\boldsymbol{a}_t|\boldsymbol{a}) = \mathcal{N}(\boldsymbol{a}, \sigma_t^2\mathbf{I})$. The final action distribution of the diffusion process is a known tractable prior distribution $\boldsymbol{a}_T \sim p_T$. An unstructured Gaussian distribution $p_T = \mathcal{N}(\mathbf{0}, \sigma_T^2\mathbf{I})$ is chosen without any information about the play data distribution.

In the case of BESO we are particularly interested in the *Probability Flow* ODE within the SDE [188]. This ODE shares the same marginal distributions $p_t(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{g})$ as the SDE at every timestep, but without the additional random noise injections. By setting $\beta(t) = 0$, we recover the *Probability Flow* ODE from Eq. (3.1):

$$d\boldsymbol{a} = -\dot{\sigma}_t\sigma_t \nabla_{\boldsymbol{a}} \log p_t(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{g})dt \tag{3.2}$$

The negative score-function $-\nabla_{\boldsymbol{a}} \log p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$ specifies the vector field of the current marginal distribution $p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$. This vector field points towards regions of low data density and is scaled with the product of the current noise level σ_t and the change of it $\dot{\sigma}_t$.

3.3.2. Diffusion Training

Algorithm 1 BESO Training

Require: Play Dataset $\mathcal{D}_{\text{play}}$
Require: Score Model $D_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, \sigma)$
Require: Noise Distribution $p_{\text{train}} = \text{LogLogistic}(P_{\text{mean}}, P_{\text{std}})$
Require: Loss weighting function $\lambda(\sigma)$
for $i \in \{1, \dots, N_{\text{train}}\}$ **do**
    Sample $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{g}) \sim \mathcal{D}_{\text{play}}$
    Sample $\sigma \sim p_{\text{train}}$
    Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$
    $\mathcal{L} \leftarrow \lambda(\sigma) \| D_\theta(\boldsymbol{a} + \boldsymbol{\epsilon}, \boldsymbol{s}, \boldsymbol{g}, \sigma) - \boldsymbol{a} \|_2^2$
    Update θ using $\nabla_\theta \mathcal{L}$
end for

In order to generate new samples by numerically approximating the reverse ODE, we require an accurate estimate of the score function $\nabla_{\boldsymbol{a}} \log p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$ for all marginal distributions p_t in our diffusion process. To achieve this, we use a neural network $D_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, \sigma_t)$ that matches the score for all marginal distributions $p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$.

$$\nabla_{\boldsymbol{a}} \log p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g}) = \left(D_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, \sigma_t) - \boldsymbol{a}\right) / \sigma_t^2. \tag{3.3}$$

The neural network is trained using the denoising score matching objective [199, 185], where we add Gaussian noise to the actions and minimize the difference between the network's output and the original actions:

$$\mathcal{L}_{D_\theta} = \mathbb{E}_{\sigma_t, \boldsymbol{a}, \boldsymbol{\epsilon}} \left[\alpha(\sigma_t) \| D_\theta(\boldsymbol{a} + \boldsymbol{\epsilon}, \boldsymbol{s}, \boldsymbol{g}, \sigma_t) - \boldsymbol{a} \|_2^2 \right], \tag{3.4}$$

where $\boldsymbol{a}$ is an action sample, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_t^2 \mathbf{I})$ represents the Gaussian noise. The losses at individual noise levels are weighted according to $\alpha(\sigma_t)$, and the current σ_t is sampled from the noise training distribution p_{train}. We use a truncated log-logistic distribution with location parameter P_{mean} and

scale parameter P_{std}: $p_{\text{train}} \sim \text{LogLogistic}(P_{\text{mean}}, P_{\text{std}})$ truncated to the range $[\sigma_{\min}, \sigma_{\max}]$. The training process is summarized in Algorithm 1. This allows us to effectively learn the noise-conditioned score function for our diffusion process and generate samples from the conditional density, $p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$, using the Probability Flow ODE.

3.3.3. Efficient Action Generation using Deterministic Samplers

Algorithm 2 Action Generation Process using DDIM based Sampler (DPM-1) adapted for BESO [116, 184]

Require: Current state $\boldsymbol{s}$, goal $\boldsymbol{g}$
Require: Score-Denoising Model $D_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, \sigma)$
Require: Decreasing noise levels $\sigma_0 > \cdots > \sigma_{N-1} > \sigma_N = 0$, with $\sigma_n = t_n$
Draw sample $\boldsymbol{a}_0 \sim \mathcal{N}(\mathbf{0}, \sigma_0^2 \mathbf{I})$
for $n \in \{0, \ldots, N-1\}$ **do**
    $\widehat{\boldsymbol{a}}_n \leftarrow D_\theta(\boldsymbol{a}_n, \boldsymbol{s}, \boldsymbol{g}, \sigma_n)$
    $\mathbf{d}_n \leftarrow (\boldsymbol{a}_n - \widehat{\boldsymbol{a}}_n)/\sigma_n$
    $\boldsymbol{a}_{n+1} \leftarrow \boldsymbol{a}_n + (\sigma_{n+1} - \sigma_n)\mathbf{d}_n$
end for
return $\boldsymbol{a}_N$

To generate actions, we sample from the prior $\boldsymbol{a}_T \sim \mathcal{N}(\mathbf{0}, \sigma_T^2 \mathbf{I})$ and numerically integrate the reverse ODE (Eq. 3.2) using our learned score model. Starting from random noise ensures diverse, multimodal action generation even though the ODE itself is deterministic—different initial samples follow different trajectories through the learned vector field.

The process begins by selecting a random sample from our prior distribution, $\boldsymbol{a}_T \sim \mathcal{N}(\mathbf{0}, \sigma_T^2 \mathbf{I})$, and then iteratively denoise this sample. Random samples as a starting point create diverse and multimodal actions, even when the underlying ODE is deterministic. The ODE can be solved numerically, by discretizing the differential equation starting from T to 0. During the action prediction, we iteratively denoise the sample at N-discrete noise levels.

BESO employs the DDIM solver, as described in detail in Algorithm 2 [115, 184], for fast, deterministic sampling. For $\sigma_t = t$, the first-order DDIM up-

date can be written as the Euler step in Algorithm 2, equivalently $\boldsymbol{a}_{n+1} = (\sigma_{n+1}/\sigma_n)\boldsymbol{a}_n + (1 - \sigma_{n+1}/\sigma_n)\widehat{\boldsymbol{a}}_n$ [184, 83]. A detailed comparison of SoTA diffusion samplers is provided in Section 3.4.2, which concludes, that DDIM has robust performance with low number of denoising steps. An additional evaluation on the influence of noise concludes that ODE solvers are competitive with SDE variants for action prediction tasks.

Our ablation studies in Section 3.4.2 verify that three denoising steps are necessary for BESO to generate actions with high accuracy to predict single actions. For the action chunking version 10 denoising steps are ideal to predict a sequence of 10 future actions. Increasing the number of inference steps further only marginally enhances the performance, while significantly slowing down the sampling process. Thus, we found that 3 and 10 steps strike the best balance between computational efficiency and performance. For inference, we can adapt the range of noise and the distribution of discrete timesteps. We retain the exponential time-step schedule and noise range $\sigma \in [0.005, 1]$ used in the published BESO recipe [162]. This default is not a claim that exponential discretization is optimal: the controlled comparison in Table 3.6 gives slightly higher mean scores for linear time steps.

3.3.4. Goal-Guided Score-based Diffusion Policies

Conditioned Policy (BESO). We define a goal-conditioned diffusion policy, $\pi(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{g})$, by directly learning the goal-and-state-conditioned distribution with our score-based generative model. In contrast to standard goal-conditioned behavior cloning, our diffusion policy allows us to capture multiple solutions present in the play data while still being expressive enough to solve long-term goals.

Goal-Classifier-Free Guided Policy (CFG-BESO). We additionally combine BESO with Classifier-Free Guidance (CFG) [67], a conditioning method for diffusion models. We train a goal-conditioned diffusion policy $\pi(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{g})$ by applying a dropout rate of 0.1 to the goal $\boldsymbol{g}$, which also trains an implicit goal-independent policy $\pi(\boldsymbol{a} \mid \boldsymbol{s})$ within our goal-conditioned model. The generation process uses a combined gradient for the denoising process

$$\begin{aligned} \nabla_a \log p_{t,\lambda}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g}) = \\ \lambda \nabla_a \log p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g}) + (1 - \lambda) \nabla_a \log p_t(\boldsymbol{a}|\boldsymbol{s}), \end{aligned} \tag{3.5}$$

where the guidance factor λ balances the influence of the goal-conditioned and goal-independent gradient. In diffusion literature, λ commonly ranges from 2 to 7.5, to guide the diffusion model towards goal-conditional distribution $\pi(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{g})$. These values result in a negative score from the goal-independent policy term. CFG has demonstrated significant performance improvements over other conditioning methods [67, 116, 138]. Even though CFG has also been successfully applied for generating state-only trajectories in Offline-RL [1], recent work on behavioral cloning suggests that CFG performs significantly worse than simpler conditioning methods [147] for step-based action generation. We provide a detailed analysis of CFG for goal-guided action generation in our experiment section and show that is an effective method to learn goal-dependent and independent policy simultaneously.

3.3.5. Diffusion Wrapper

One of the main challenges of training the score-based diffusion model is the big range of noise levels $\sigma_t \in [0.001, 80]$ To address this challenge, we use an improved architecture [83] including additional skip-connections and two pre-conditioning layers, which are conditioned on the current noise level σ_t

$$\begin{aligned} D_\theta(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g}, \sigma_t) = \\ c_{\text{skip}}(\sigma_t)\boldsymbol{a} + c_{\text{out}}(\sigma_t)F_\theta(c_{\text{in}}(\sigma_t)\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, c_{\text{noise}}(\sigma_t)), \end{aligned} \tag{3.6}$$

Details for each preconditioning function are provided in Section 2.1.3. The diffusion wrapper is also visualized in the middle of Figure 3.1. These additional skip connections help the score model to scale the output to a wide range of noise levels σ_t, either by estimating the denoised sample $\boldsymbol{a}_{t-1}$, directly predicting the noise ϵ or something in between these two objectives.

3.3.6. Model Architecture

In this section, we introduce two variants of BESO optimized for synthesizing actions for goal-conditioned behavior. BESO uses a Transformer-based architecture with causal masking as the inner model $F_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, \sigma_t)$. This architecture enables BESO to learn temporal relations between observations and actions, thereby improving its overall performance. A detailed overview

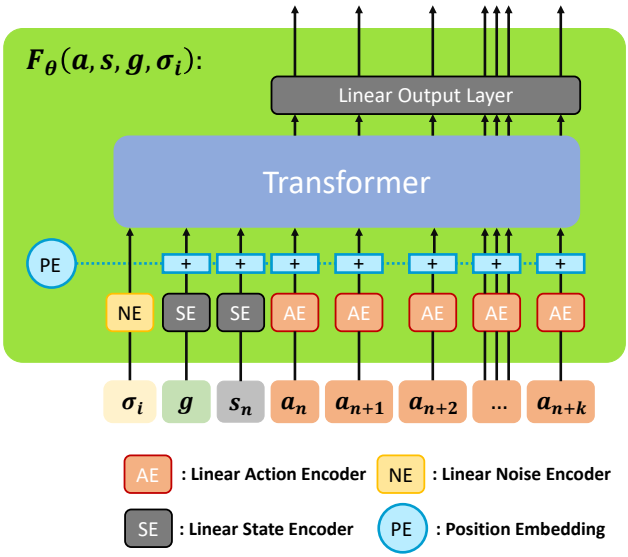


Figure 3.2.: Visualization of the Seq-BESO model architecture. Seq-BESO predicts a sequence of future actions conditioned on the current state token and desired goal with causal masking.

of our proposed architecture is shown in Figure 3.1. Three linear embedding layers encode the states s_n, noise σ_t and the noisy actions a_n into a linear representation of the same dimension, $l_s(s), l_a(a), l_\sigma(\sigma)$. In addition, the position embedding information is added to the linear representations. The noise embedding is concatenated with the desired future states and all state-noise-action pairs in a large sequence for the model. To take advantage of the causal masking in the transformer, we concatenate the goal sequence before the current observation sequence [31], allowing for a sequence of goal states. All variants of BESO are flexible enough to encode either future state goals or language-based objectives. This is achieved through the integration of a pre-trained CLIP Text model [156], which transforms a sentence into a corresponding latent vector, thereby enabling learning from text instructions. To encode images, we leverage ResNets-18 with spatial softmax pooling

		GCBC	C-IBC	LMP	RIL	C-BeT	CX-Diff	BESO	CFG-BESO	Seq-BESO
Block-Push	Rew.	0.13 (± 0.04)	0.46 (± 0.06)	0.04 (± 0.03)	0.06 (± 0.01)	0.91 (± 0.03)	0.93 (± 0.03)	*0.96* (± 0.02)	***0.97*** (± 0.02)	0.95 (±0.03)
	Res.	0.13 (± 0.04)	0.29 (± 0.10)	0.04 (± 0.03)	0.02 (± 0.01)	0.87 (± 0.07)	0.90 (± 0.04)	***0.93*** (± 0.02)	0.88 (± 0.04)	0.92 (±0.03)
Relay-Kitchen	Rew.	2.65 (± 0.25)	0.50 (± 0.09)	1.45 (± 0.22)	0.31 (± 0.15)	2.73 (± 0.28)	3.64 (± 0.14)	***3.98*** (± 0.07)	***3.98*** (± 0.07)	3.59 (±0.08)
	Res.	2.57 (± 0.26)	0.45 (± 0.08)	1.41 (± 0.22)	0.23 (± 0.11)	2.69 (± 0.28)	3.35 (± 0.15)	***3.75*** (± 0.08)	*3.47* (± 0.08)	3.33 (± 0.09)

Table 3.1.: Average performance of policies on the conditioned block-push and kitchen environment. Results are averaged over 10 seeds with 100 rollouts each. Rew. denotes total reward for completing any task; Res. denotes completion of the specified goals. The latent-plan baseline is LMP. BESO and CFG-BESO consistently outperformed all baselines, despite using only 3 inference steps. Both variants of BESO show a low deviation across seeds, indicating their robustness.

instead of global average pooling and Group Norm [27]. For our language-conditioned experiments, we use additional FiLM-conditioning [149] for the ResNets to condition the latent representation in the ResNet on the language instruction by using feature-wise modulation. We present two variants of our proposed model architecture:

- **BESO:** an autoregressive-Decoder architecture, that predicts a single action at every timestep. The default architecture is visualized on the right part of Figure 3.1.
- **Seq-BESO:** Inspired by the action chunking concept from [222, 27], this variant forecasts the subsequent H actions in a single stride. The depiction of this action chunking decoder architecture is available in Figure 3.2.

Our empirical observations indicate that both models exhibit particular strengths and weaknesses contingent upon the environment's specific requirements.

3.4. Evaluation

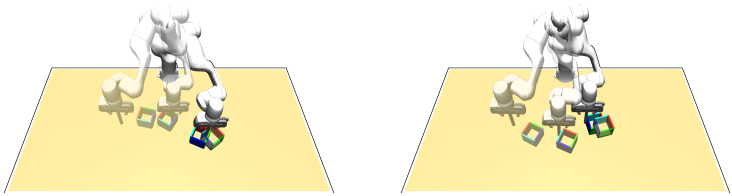
The objective of our experiments was to answer the following key questions:

- (I) Is BESO competitive on goal-conditioned environments against SoTA baselines?

- (II) Does Classifier-Free Guidance work for goal-conditional behavior synthesis?
- (III) How effectively does BESO learn from limited demonstrations?
- (IV) How does our method perform in challenging long-horizon settings?
- (V) How good is BESO in capturing the multimodality in human behavior?
- (VI) What are the key components to enable fast sampling of Diffusion policies with good performance?

To answer these questions, we evaluated BESO and its variants CFG-BESO and Seq-BESO in five different challenging simulation scenarios with various ablation experiments. First, we introduce the baselines, that are compared against different BESO versions. Afterward, we will discuss every research question with related experiments to answer them.

3.4.1. Comparative Evaluation



[a] Push from inside [b] Push from outside

Figure 3.3.: Overview of the Aligning task for testing the multimodality of different policies. The robot is tasked with pushing the box to a randomly generated target position. The dataset contains 1000 human recorded demonstrations with 500 trajectories for each behavior.

	Aligning	
	Success Rate	Entropy
BC-MLP	0.708 (± 0.052)	0.0 (± 0.0)
IBC	0.638 (± 0.027)	0.300 (± 0.048)
BeT	0.645 (± 0.069)	0.536 (± 0.105)
BESO	**0.861** (± **0.016**)	**0.711** (± **0.030**)

Table 3.2.: Evaluation of different policy representations on the Aligning Task. We report the average success rate and a measurement of the Entropy, that measures the multimodality on different methods. BESO achieves the highest success rate and entropy.

3.4.1.1. State-based Simulation Experiments.

First, BESO, CFG-BESO, and Seq-BESO are tested against various methods on two established benchmarks for imitation learning: the **Block-Push Environment** [47] and the **Relay Kitchen Environment** [57]. We used the adapted goal-conditioned variant [31]. Both environments are depicted in Figure 2.4 and Figure 2.5 and described in Section 2.3.5.2 and Section 2.3.5.1. We used the same experiment settings as described in [31] to allow for fair comparisons. The methods were evaluated on two metrics in the two environments: **result** evaluates how many of the desired goals of each rollout are achieved, while **reward** measures the overall performance by giving credit for reaching any goal defined in the environment.

Baselines. We compare BESO against several SoTA methods:

Goal-conditioned Behavior Cloning (GCBC) learns an unimodal policy encoded as an MLP trained with an MSE loss [119].

Relay Imitation Learning (RIL) is a hierarchical policy, that learns a high-level sub-goal generator, which is used to condition a low-level policy [57].

Latent Motor Plans (LMP & **HULC)** are hierarchical goal-conditioned policies, which consist of a seq2seq CVAE and an action decoder policy [119]. For state-based and image-only environments we use the LMP baseline [119]. For the challenging CALVIN benchmark, we use HULC, which has an adapted KL-weighting term and a transformer encoder with a discrete latent planning space optimized for this setting [129].

Conditional Implicit Behavior Cloning (C-IBC) uses an energy-based model as an implicit policy [47]. We use a goal-conditioned extension of IBC to study the importance of the selected generative model architecture to learn multimodal behavior.

Conditional-Behavior Transformer (C-BeT) is a GPT-like transformer-based policy, that predicts discrete action labels together with a continuous offset vector to learn multimodal behavior [172, 31]. The action labels are determined a priori via K-means clustering.

Diffusion-X (CX-Diff) [147] is a DDPM [66] based policy with improved inference. It uses stochastic sampling and additional X-extra inference steps at the lowest noise level to synthesize actions in 50+X steps. While performing only slightly worse than the closely related KDE-Diff [147] it has a significantly lower computational cost.

To ensure a fair evaluation of all methods we kept the general hyperparameters, e.g., layer size and number, as consistent as possible while tuning the method-specific hyperparameters. Additionally, we evaluated all models on the kitchen and block-push task with 10 seeds and 100 rollouts each. Given the high computational costs and time of training models for CALVIN and the Alignment task, we restricted the tested methods to 3 seeds and limited the most competitive methods to these challenges.

Results. The results are summarized in Table 3.1. BESO consistently outperforms all baselines on both tasks across 10 seeds. The low variance of BESO, additionally, indicates the robustness of our approach. In these experiments, BESO outperforms its action chunking Seq-BESO variant in all metrics. Among the baselines, Diffusion-X and C-BeT perform well on the kitchen task and block-push environment, respectively.

These results are particularly notable given that BESO uses only 3 denoising steps, compared to CX-Diff which requires 50 stochastic steps plus 8 additional refinement steps at the lowest noise level. By contrast, CX-Diff, when limited to 3 denoising steps, only managed an average result of 2.74($\pm$0.26) in the kitchen environment. On a modern desktop PC, BESO requires around 0.012 seconds to predict an action, while the CX-Diffusion model needs an average of 0.15 seconds. BESO is 10 times faster. Overall, our results indicate that all BESO variants are competitive against SoTA baselines and capable of effectively learning from play data, making it a promising approach for goal-conditioned behavior learning.

3.4.1.2. Multimodal Behavior Experiment

After verifying the general performance of score-based diffusion policies in state-based settings, we want to evaluate the multimodality of diffusion policies in a separate experiment.

Prior tasks in the relay kitchen and block push setting already showed the ability of BESO and Seq-BESO to capture multimodal behavior. However, in these goal-conditioned settings, it is difficult to analyze the ability of score-based diffusion policies to capture the diversity of human demonstrations. Thus, we conduct a separate experiment in the Aligning task from the D3IL benchmark [78]. The task requires agents to navigate a box to a target location with two possible ways to solve the task. The dataset includes 500 demonstrations for each solution mode, employing the entropy metric to compare the multimodality of score-based diffusion policies against baseline models. The task and its two distinct behaviors to solve it are visualized in Figure 3.3. Agents get a full credit for successfully reaching the target position. In this experiment, we compare BESO against IBC and BeT, two recent frameworks, that report strong performance in learning multimodal behavior.

Results, summarized in Table 3.2, highlight BESO's superiority in both average success rate and entropy, evidencing its proficiency in learning multimodal behavior. While IBC and BeT also capture multimodal behavior, their success rates lag behind the standard BC baseline. Conversely, BESO not only augments average performance by 20% but also ensures high behavioral diversity. These results address Research Question (**V**), demonstrating that **diffusion policies capture diverse behavior better than alternative policy representations without sacrificing performance.**

Combined with the state-based experiments in Section 3.4.1.1, we can answer Question (**I**) in the affirmative: BESO surpasses current SoTA baselines on goal-conditioned environments.

3.4.1.3. Performance on low-data regimes

We test the ability of BESO to learn from few demonstrations on the Aligning task from the multimodal experiment. The results of this experiment are summarized in Figure 3.4. BESO achieves the highest success rate and entropy

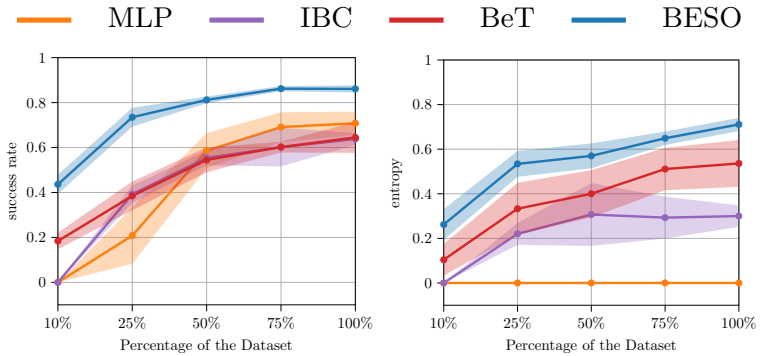


Figure 3.4.: Ablation study on dataset size: performance on the Aligning task when training on different fractions of the dataset. BESO outperforms all other policy representations in all dataset sizes in success rate and entropy of learned solutions. In addition, BESO has the lowest standard deviations.

on all dataset splits and shows strong performance even when trained on only 10% of the full dataset. Moreover, BESO exhibits the most diverse behaviors across all smaller dataset splits. These results answer Question (**III**): **score-based diffusion policies learn effectively from small datasets.**

3.4.1.4. Image-based Goal-Conditioned Experiments

Before scaling to language-conditioned multi-task learning, we evaluate BESO variants on image-based goal conditioning using the CALVIN image-only benchmark [167]. The dataset comprises 6 hours of unstructured play data with a single static RGB camera. Crucially, goal images are sampled from states not seen during training, with the end-effector positioned away from the task-relevant objects—requiring policies to infer the necessary state changes rather than simply imitating end-effector positions.

Figure 3.5 summarizes results on 13 individual tasks and 2-task sequences. Seq-BESO achieves the best performance in both settings, with the autoregressive BESO variants ranking second. The performance gap between Seq-BESO and autoregressive BESO is substantially larger in image-based settings than in the state-based experiments of Section 3.4.1.1. We attribute this to action chunking providing temporal consistency that compensates for the partial

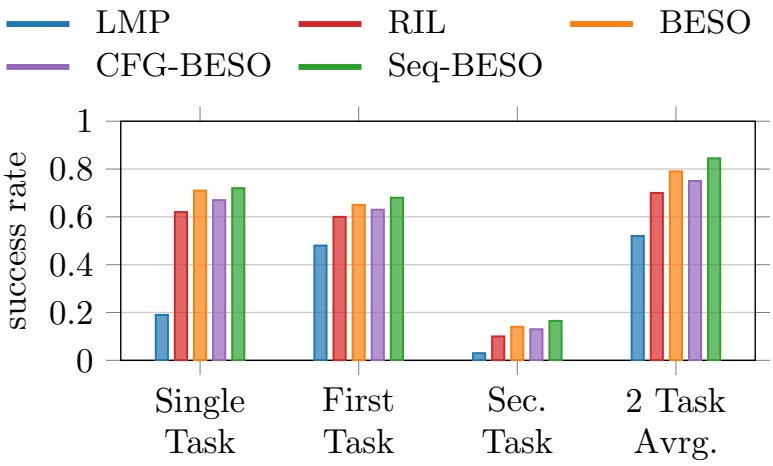


Figure 3.5.: Performance on the CALVIN image-only benchmark. Left: average success rate on 13 individual tasks. Right: success rate when conditioned on a single goal image requiring completion of 2 sequential tasks.

observability inherent in image conditioning. Based on these findings, subsequent image-based experiments use Seq-BESO.

3.4.1.5. Language-Conditioned Multi-Task Experiments

Next, we evaluate Seq-BESO on the CALVIN benchmark [130] to assess long-horizon task completion, zero-shot generalization, and scaling with dataset size. We compare against HULC [129], the prior SoTA, and C-BeT [31] re-implemented with identical encoders for fair comparison. Details on baselines and hyperparameters are provided in Appendix A.2.

Results. Table 3.3 summarizes results across three CALVIN settings.

Single-environment training (D→D). Seq-BESO achieves an average rollout length of 3.19, a 19% improvement over HULC (2.68).

Zero-shot generalization (ABC→D). When tested on environment D after training only on A, B, and C, Seq-BESO maintains an average rollout length of 2.06—a relative drop of only 35% from the D→D setting. In contrast, HULC

Train→Test	Method	No. Instructions in a Row (1000 chains)					
		1	2	3	4	5	**Avg. Len.**
D→D	HULC	82.5%	66.8%	52.0%	39.3%	27.5%	2.68±(0.11)
	C-BeT	62.6%	30.0%	12.5%	4.3%	1.6%	1.11±(0.15)
	Seq-BESO	**89.4%**	**74.0%**	**61.1%**	**51.7%**	**42.6%**	**3.19±(0.06)**
ABC→D	HULC	41.8%	16.5%	5.7%	1.9%	1.3%	0.67±(0.10)
	C-BeT	44.6%	16.4%	5.5%	2.0%	0.2%	0.69±(0.02)
	Seq-BESO	**76.9%**	**53.3%**	**35.8%**	**24.2%**	**16.1%**	**2.06±(0.13)**
ABCD→D	HULC	88.9%	73.3%	58.7%	47.5%	38.3%	3.06±(0.07)
	C-BeT	80.3%	54.0%	32.0%	18.8%	10.3%	1.94±(0.08)
	Seq-BESO	**93.6%**	**87.3%**	**81.0%**	**74.9%**	**67.7%**	**4.05±(0.04)**

Table 3.3.: Performance of Seq-BESO and baselines on the CALVIN benchmark. We evaluate on three settings: single-environment training (D→D), zero-shot generalization (ABC→D), and large-scale multi-environment training (ABCD→D). All methods are tested on environment D following the protocol of [130]. Results are averaged over 1000 rollouts with instruction chains of 5 tasks and 3 random seeds.

drops by 75%, struggling to complete even one task on average. This demonstrates that diffusion policies learn more transferable representations.

Large-scale training (ABCD→D). With access to all four environments (22,966 sequences), Seq-BESO achieves 4.05 average rollout length, outperforming HULC by 32%. Notably, Seq-BESO benefits more from additional data: performance improves by 27% from D-only to ABCD training, compared to 14% for HULC.

These results answer Question (**IV**), demonstrating that score-based diffusion policies excel at long-horizon tasks, generalize effectively to novel scenes, and scale efficiently with dataset size.

3.4.1.6. Real World Play Experiments

Finally, we evaluate BESO in a challenging real-world play kitchen setup across 10 different tasks collected by human experts. This evaluation aims to assess BESO's suitability as a policy for real-world applications. Detailed overview of the data collection and kitchen setup is summarized in Section 2.3.5.6.

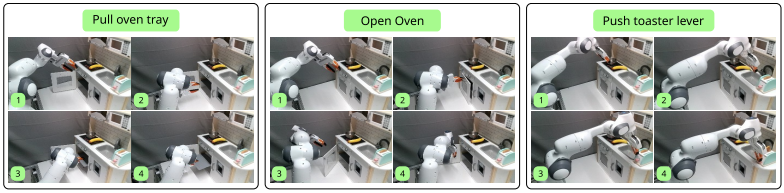


Figure 3.6.: Visualization of Seq-BESO rollouts in the play kitchen. The rollouts demonstrate BESO's capability to handle complex manipulation tasks involving articulated objects and requiring precise actions. BESO can efficiently perform a diverse set of tasks with only a few demonstrations.

Dataset. The evaluation encompasses 10 distinct tasks from the 20-task repertoire of our real-world play kitchen. We use the same play-data collection as the FLOWER experiments in Chapter 6; the chapters evaluate different task subsets. Data collection involved three human experts who generated long-horizon play data through leader–follower teleoperation, kinesthetically guiding a second Panda robot as the leader. To ensure diversity, data collectors were instructed to complete tasks in arbitrary order, resulting in 45 minutes of play data.

Next, we segment and annotate short-horizon sequences from the dataset with language descriptions of the tasks. The final language-annotated dataset comprises 417 short-horizon sequences of language-annotated trajectories. This shared 417-segment, 45-minute collection is used for BESO's ten-task evaluation here and FLOWER's twenty-task evaluation in Chapter 6. MDT uses the separate 4.5-hour collection described in Section 4.4.3. The policies learn to predict a sequence of 10 future joint states conditioned on two camera images at 6Hz. Given the relatively small number of individual trajectories for each task compared to related real-world datasets, we co-trained our policies on $1k$ trajectories from the large-scale diverse DROID dataset [85]. Samples from our dataset and DROID were weighted equally at 0.5 to ensure that the policy had an equal likelihood of encountering samples from both our kitchen play dataset and DROID.

An overview of 3 tasks used for evaluation is provided in Figure 3.6 while the remaining 7 tasks are visualized in Figure A.1 in the Appendix. For evaluation each policy is tested 10 times from a starting position not seen in training with some added noise to it.

During our experiments, we further varied the orientation of the banana slightly for the robot to pick up, while we kept the toaster in the same position during all our experiments. Policy training was conducted on real robot data for 24 hours using 4 GPUs for $100k$ steps. Checkpoint selection was based on the lowest validation loss to guarantee a fair comparison.

Baselines. We compare SEQ-BESO against a transformer-based GCBC baseline that employs a GPT-based Transformer Policy with action chunking without diffusion. Both models use a history length of 1 and predict 10 actions. This comparison aims to investigate the impact of using a diffusion-based policy objective. To ensure a fair comparison, identical hyperparameters and training duration is applied for both models.

Task	GCBC	Seq-BESO
Pot from right stove to sink	40%	100%
Pot from sink to right stove	0%	70%
Open oven	10%	100%
Pull oven tray	0%	100%
Open microwave	40%	100%
Close microwave	20%	90%
Banana from right stove to sink	20%	90%
Banana from sink to right stove	0%	90%
Push toaster lever	0%	60%
Pickup toast and put to sink	0%	40%
Mean	13%	84%

Table 3.4.: Evaluation results for play kitchen for Seq-BESO and GCBC. The results are averaged over 10 rollouts each.

Results. We present the average results across all tasks, each averaged over 10 rollouts, in Table 3.4. Our findings demonstrate that BESO outperforms the GCBC policy by a wide margin and achieves a high success rate across all tasks. These results reinforce our simulation findings and provide strong evidence that BESO is an effective policy for real-world manipulation. We additionally provide videos of policy rollouts on our project webpage.

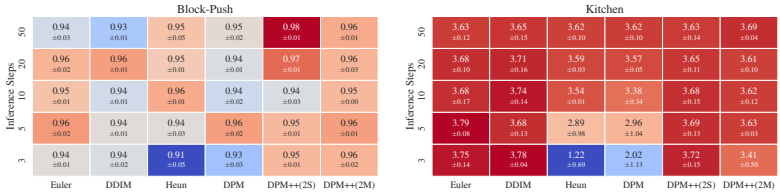


Figure 3.7.: Ablation study for different deterministic solvers for BESO. Results are reported on the relay kitchen and block-push environment and averaged over 100 rollouts with 3 seeds each.

3.4.2. Ablations

This section covers key ablations of BESO to analyze various components and their impact to the final performance.

3.4.2.1. Classifier-Free Guidance Experiments

We address Question (**II**) by evaluating the effect of Classifier Free Guidance (CFG) for action generation with goal-conditioned policies in the autoregressive step-based setting. CFG-BESO uses dropout on the goal during training to learn a goal-dependent score and independent score together. During test time the parameter λ controls, how the joint score of both these models is composed to diffuse actions. For $\lambda = 0$, the model predicts actions with its goal-independent policy only, and for values $\lambda \geq 1$, the score of the goal-conditioned policy has a higher influence on the joint score.

We test CFG-BESO with different values of λ, to study the performance of CFG in two environments. Figure 3.8 demonstrates that CFG effectively guides goal-conditioned behavior. The average result for optimal λ values for the block-push and kitchen tasks is slightly worse than the standard goal-conditioned variant, while the average reward is equal. The performance of the CFG-model with $\lambda = 0$, which results in the model omitting all goal-conditioning demonstrates, that CFG-BESO is capable of learning a well-performing, unconditional policy $\pi(\boldsymbol{a}|\boldsymbol{s})$.

The low average result in Figure 3.8 shows that the policy ignores the goal state and aims to achieve a high reward solely based on the current state. By

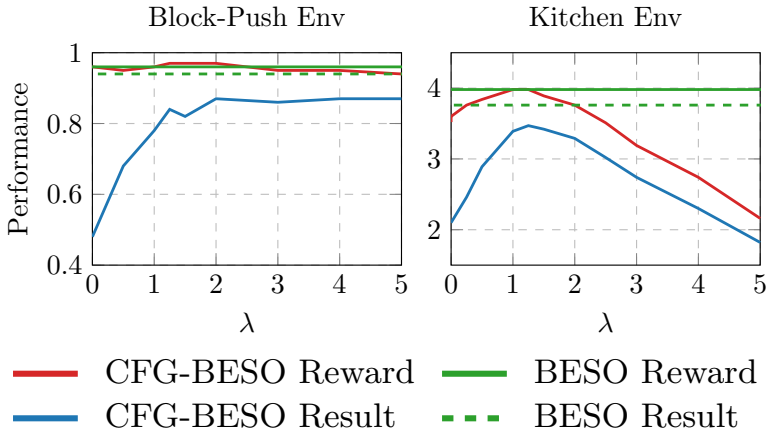


Figure 3.8.: Comparison of CFG Method for GCIL from Play Data. For CFG-BESO we evaluate the 10 seeds on 100 rollouts each with different λ values. The CFG variant of BESO has a slightly worse average result in both environments with similar rewards. When using $\lambda = 0$, we can recover an unconditional policy, that performs random rollouts with high rewards and low results.

increasing the CFG parameter λ the model shifts its behavior towards solving defined goals. This gives CFG-BESO a unique advantage over common play-based policies. However, CFG has a trade-off: it slightly lowers the average result for more diverse rollouts.

Empirical evaluations suggest the best λ value is 1.25 for most tested environments. In this setting, the score used in the action generation process is composed of two scores: the unconditional one and the goal-conditioned variant. These results demonstrate that several diffusion policy scores can be composed together during inference and achieve competitive results. Experiments with higher values resulted in a lower average performance in environments with high-dimensional action spaces, indicating instability in the action generation. We hypothesize that the guidance provided by the goal-conditioning is only crucial in certain steps during the rollouts, specifically when the policy is deciding which task to solve.

3.4.2.2. BESO Policy Ablations

We answer Question (**VI**) by evaluating different components of BESO to study their contribution to the overall performance.

Conditioning Method. First, we evaluated different methods to condition the behavior generation on the desired goal state. We tested the FiLM-conditioning [149] and the sequential conditioning method used in C-BeT [31]. FiLM requires additional MLP models, which input the goal and scale the latent representations inside the transformer layers. The sequential conditioning method simply includes desired goal-states at the beginning of our sequence as depicted in the model overview of Figure 3.1. We tested both conditioning variants using the same transformer score model and evaluated it on the block-push and kitchen environment on 10 seeds. FiLM conditioning resulted in a performance drop compared to the sequential conditioning method from an average result of 0.93 to 0.91 and 3.76 to 3.4 on the block-push and kitchen environment respectively. Moreover, the FiLM method increases the overall model capacity. Hence, BESO uses the sequential conditioning method.

Sampling Algorithm. BESO generates actions by numerically approximating the reverse ODE with its learned score-model starting from a sample generated from our Gaussian prior distribution p_T. We investigated several numerical sampling algorithms used in diffusion research, such as DDIM [184], DPM [115], DPM++ [115], and Heun [83], to assess their contribution to BESO's performance. The samplers were evaluated on the block-push and kitchen environments with a different number of denoising steps. The results show that the performance gap between the individual samplers is small, with DDIM achieving the best average performance for a low number of inference steps. DPM++(2S) a variant of the DPM solver achieves the second best performance on average [116]. Surprisingly, the second-order Heun solver has the worst average performance. Overall BESO is robust to the number of sampling steps and chosen sampler type, maintaining a similar performance from 3 to 50 inference steps. Given the increase in computational cost during inference we use 3 denoising steps in our experiments.

Stochastic vs. Deterministic Sampling. Current diffusion literature supports the assumption that stochastic samplers have a better overall performance compared to deterministic samplers [83, 188]. We tested this assumption with respect to step-based action generation. We evaluated the same

		Deterministic	Stochastic
Block-Push	Reward	0.97 (± 0.02)	0.97 (± 0.02)
	Result	**0.93** (± **0.02**)	0.92 (± 0.03)
Relay-Kitchen	Reward	3.95 (± 0.10)	**4.03** (± **0.07**)
	Result	3.73 (± 0.11)	**3.80** (± **0.08**)
CALVIN	Hard Tasks	**0.71** (± **0.01**)	0.68 (± 0.03)
	2 Tasks	0.79 (± 0.03)	0.79 (± 0.02)

Table 3.5.: Evaluation of the Influence of Noise Injection for the performance of BESO. Results are averaged over 2 samplers with and without random noise injection.

models with 2 sampling algorithms DPM++(2S) and the Euler sampler [116, 83], each with and without noise injection. The noise scheduling was performed via the ancestral sampling strategy, as used in the DDPM variant [66, 188]. Experiments were again conducted in three environments. As shown in Table 3.5, the results suggest that the addition of noise does not offer a significant benefit to the action generation of step-based diffusion policies. Stochastic samplers only increase the average performance in the kitchen environment. The discrepancy compared to common diffusion applications such as image synthesis [34] could be rooted in high-dimensional image spaces, making the generation process more difficult and requiring more steps for good results. In these high-dimensional spaces, errors are more likely to occur and accumulate over time. Adding noise during the inference process helps the model to correct errors of the gradient approximation, resulting in a better overall performance [83]. In contrast, step-based action-distributions are significantly lower dimensional than the high-dimensional latent spaces of image generation, hence, the addition of noise does not appear to benefit the average performance of step-based policies, as supported by our experimental results.

Discretization of Time Steps. Next, we analyze the importance of the time steps discretization for inference of score-based diffusion policies. The time steps function determines how noise levels are distributed over the discrete steps. Our empirical evaluation summarized in Table 3.6. Linear time steps achieve the highest reported mean scores in this comparison; Karras and exponential discretizations produce similar scores. The exponential schedule remains the default of the published recipe described in Section 3.3.3,

Sampler Type	Block-Push		Relay Kitchen	
	Reward	Result	Reward	Result
Exponential	0.96 (± 0.02)	0.93 (± 0.02)	3.86 (± 0.09)	3.74 (± 0.09)
Linear	**0.97 (± 0.01)**	**0.94 (± 0.01)**	**3.98 (± 0.08)**	**3.76 (± 0.10)**
IDDPM	0.95 (± 0.01)	0.92 (± 0.01)	3.88 (± 0.08)	3.67 (± 0.09)
Karras	0.96 (± 0.02)	0.93 (± 0.03)	3.96 (± 0.08)	3.75 (± 0.07)
VE	0.95 (± 0.01)	0.93 (± 0.01)	3.98 (± 0.09)	3.75 (± 0.10)
VP	0.64 (± 0.17)	0.61 (± 0.16)	3.21 (± 0.12)	2.95 (± 0.14)

Table 3.6.: Ablation for different time steps for denoising of BESO. Results are averaged over 10 seeds and 100 rollouts each. All models use the DDIM solver with 3 denoising steps.

while this ablation motivates checking linear discretization for a new environment.

3.4.2.3. Summary of Key-Findings

Across 54 tasks in 5 environments, BESO demonstrates three key properties:

1. **Expressiveness and Multimodality:** BESO captures multimodal behavior without sacrificing accuracy, outperforming IBC and BeT on entropy while achieving 20% higher success rates.
2. **Scalability:** Performance scales with dataset size (27% improvement from D to ABCD on CALVIN) and generalizes to unseen environments (35% relative drop vs 75% for HULC).
3. **Efficiency:** Three denoising steps suffice for state-based tasks, achieving 10× faster inference than CX-Diff.

For image-based settings, Seq-BESO with action chunking is recommended; for state-based settings, autoregressive BESO performs best.

3.4.3. Limitations

BESO shows strong performance in different environment settings in learning goal-conditioned behavior from future desired states, images, and language. However, several limitations remain:

1. **Training Speed.** While BESO and other Diffusion Policies train significantly faster than Energy-based Models, their training speed is still slower compared to default Behavior Cloning baselines.
2. **Inference Speed.** Diffusion Policy inference still requires several denoising steps in a forward pass to generate the next actions. While our improvements lead to a significant speed-up compared to concurrent diffusion policy designs, the lower speed compared to explicit policies remains. However, recent work in Consistency Models [187] and Score-Distillation methods [168] show promising results to reduce the number of denoising steps to a single forward pass.
3. **Multimodal Goals.** Our proposed architecture is only able to learn from either language goals or image goals and cannot learn from multiple goal modalities simultaneously.
4. **Online Learning.** BESO, in its current form, is unable to utilize new samples to improve performance post-training. The score-matching objective requires recorded demonstrations and cannot be inferred from a reward, as this would require differentiation through the complete score-ODE. The problem of differentiation through the complete Score-ODE is a challenging and active area of research [197].

Future work could explore ways to incorporate recent theoretical findings to develop new algorithms for diffusion policies to leverage new samples and adapt its policy on the fly.

3.5. Conclusion

We introduce BESO a novel policy representation for goal-conditioned imitation learning, which leverages score-based diffusion models to generate actions. We use the expressiveness and multimodal properties of score-based diffusion models to learn task-agnostic behavior from offline, reward-free play datasets, without requiring hierarchical structures or additional clustering. In addition, we demonstrated the effectiveness of Classifier-Free Guidance for simultaneously learning a goal-dependent and goal-independent policy in a sequential setting.

Experiments on several challenging GCIL benchmarks showed that BESO significantly improves upon several SoTA GCIL algorithms. Our ablation studies have demonstrated the key components of BESO that enable fast, deterministic behavior generation and detailed analysis to study the multimodality aspect of score-based diffusion policies. It further outperformed standard DDPM policies with only 3 denoising steps, alleviating prior drawbacks of slow diffusion sampling.

While BESO demonstrates great performance as a standalone policy, it also offers the flexibility to be seamlessly integrated into other hierarchical frameworks as an action prediction policy.

4. Multimodal Diffusion Transformer: Learning Versatile Behavior from Multimodal Goals

Building on the score-based diffusion policy from Chapter 3, this chapter addresses challenge **C2**: learning from partially labeled play data. While BESO established an expressive policy backbone, it relies on unimodal goal definitions and cannot leverage datasets where only a fraction of demonstrations contain language annotations. MDT overcomes this limitation through two self-supervised objectives: CLA aligns goal-conditioned representations across modalities using contrastive learning, while MGF enforces that the policy's latent space encodes sufficient information to predict future visual states. Together with an improved transformer architecture, these objectives enable MDT to achieve state-of-the-art performance while requiring less than 10% of the parameters of competing methods.

The content of this chapter is based on the paper ***Multimodal Diffusion Transformer: Learning Versatile Behavior from Multimodal Goals*** *(Moritz Reuss, Ömer Erdinç Yağmurlu, Fabian Wenzel, Rudolf Lioutikov, 2024), published in* ***Robotics: Science and Systems (RSS)****. Wording, notation, and formulations were revised for this thesis.*

4.1. Introduction

Future robot agents need the ability to exhibit desired behavior according to intuitive instructions, similar to how humans interpret language or visual cues to understand tasks. Current methods, however, restrict conditioning

on either language instructions [180, 208] or visual goals [31, 162]. This restriction limits training to fully-labeled datasets, which is not scalable for creating versatile robotic agents.

Natural language commands offer the greatest flexibility to instruct robots, but training language-conditioned policies requires large, fully-annotated datasets that are time-consuming to collect [117, 118]. In contrast, LfP datasets are fast to collect but contain sparse language annotations [119, 130]. Existing methods cannot efficiently leverage such partially-labeled data.

This chapter introduces MDT, a diffusion-based approach that learns from multimodal goals—both language and images—simultaneously. Building on the score-based diffusion policy from Chapter 3, MDT adds two self-supervised objectives to handle sparse language labels: CLA aligns goal-conditioned representations across modalities using contrastive learning, while MGF enforces that the policy's latent space encodes sufficient information to predict future visual states. These auxiliary losses improve performance without additional inference cost.

MDT achieves SoTA results on CALVIN [130], a popular benchmark for language-guided learning from play data comprised of human demonstrations with few language annotations. On CALVIN ABCD→D, MDT-V increases the average rollout length from RoboFlamingo's 4.09 to 4.52, a gain of 0.43 completed tasks (10.5% relative), with fewer than 10% of its trainable parameters and without additional policy pretraining. The results show that MDT is even competitive to state-of-the-art methods, that are trained on the fully annotated dataset. Through a series of experiments and ablations, the efficiency of the method and the strategic design choices are thoroughly evaluated. The major contributions of this chapter are threefold:

- We introduce Multimodal Diffusion Transformer, a novel Transformer-based Diffusion approach. MDT excels in learning from multimodal goals, sets a new state-of-the-art performance on the CALVIN Challenge, and achieves strong results on LIBERO with sparse language annotations.
- Two simple yet effective self-supervised losses for visuomotor policies to improve learning from multimodal goals. MGF and CLA improve the performance of multi-task behavior learning from sparsely labeled datasets without additional inference costs.

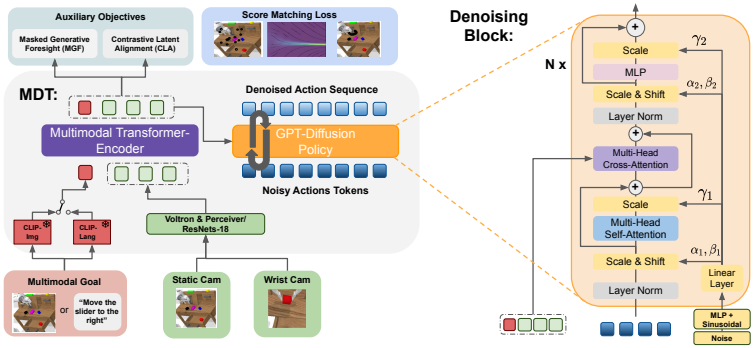


Figure 4.1.: Overview of MDT. (Left) Overview of the proposed multimodal Transformer-Encoder-Decoder Diffusion Policy used in MDT. (Right) Specialized Diffusion Transformer Block for the Denoising of the Action Sequence. MDT learns a goal-conditioned latent state representation from multiple image observations and multimodal goals. The camera images are processed either via frozen Voltron Encoders with a Perceiver or ResNets. The separate GPT denoising module iteratively denoises an action sequence of 10 steps with a Transformer Decoder with Causal Attention. It consists of several Denoising Blocks, as visualized on the right side. These blocks process noisy action tokens with self-attention and fuse the conditioning information from the latent state representation via cross-attention. MDT applies adaLN conditioning [148] to condition the blocks on the current noise level. In addition, it aligns the latent representation tokens of the same state with different goal specifications using self-supervised contrastive learning. The latent representation tokens are also used as a context input for the masked Image Decoder module to reconstruct masked-out patches from future images.

- A comprehensive empirical study covering 184 tasks (34 in CALVIN, 130 in LIBERO, and 20 on the real robot) across several benchmarks, verifying the performance and effectiveness of MGF and MDT.

4.2. Related Work

Language-Conditioned Robot Learning Language is an intuitive and understandable interface for human-robot interactions with active research for language-guided learning methods within the robotics community. A growing body of work uses these models as feature generators for vision

and language abstractions for downstream policy learning [180, 127, 9, 129, 118, 25, 210] and improved language expression-grounding [52, 74, 159, 212, 189]. Notably, methods like CLIPort [179] employ frozen CLIP embeddings for language-guided pick and place, while others, such as PaLM-E [39] and RoboFlamingo [105], fine-tune vision-language models for robot control. Other methods focus on hierarchical skill learning for language-guided manipulation in LfP [130, 119, 127, 167, 223, 202]. Further, transformer-based methods without hierarchical structures [226, 31, 162, 208, 172, 170], focus on next-action prediction based on previous observation tokens. MT-ACT, for instance, utilizes a VAE transformer encoder-decoder policy, encoding only the current state and a language goal to generate future actions [9, 49, 222].

Furthermore, diffusion-based methods have gained adoption as policy representations that iteratively diffuse actions from Gaussian noise [66, 188]. Several diffusion policy approaches focus on generating plans on different abstraction levels for behavior generation. LCD [220] trains a diffusion model to diffuse a latent plan sequence in pre-trained latent spaces of HULC [129] combined with HULC's low-level policy. UniPi [41] and AVDC [92] directly plan in the image space using video diffusion models and execute the plan with another model. Frameworks related to MDT are Distill-Down [58] and Play-Fusion [25], which also utilize diffusion policies for language-guided policy learning. Both methods use variants of the CNN-based diffusion policy [27]. However, all these methods require fully annotated datasets to learn language-conditioned policies. MDT effectively learns from multimodal goals, enabling it to leverage partially annotated datasets.

Self-Supervised Learning in Robotics An increasing body of work in robotics studies self-supervised representations for robot control. A key area is learning universal vision representations or world-models, typically trained on large, diverse offline datasets using either masking strategies [82, 63, 51, 213, 171, 124] or contrastive objectives [55, 143, 173, 218, 136, 95, 6, 123, 158]. Another body of work explores robust representations for robot policies from multiple sensors, using token masking strategies [157] or generative video generation [208]. However, these methods require specific transformer models that rely on a long history of multiple states, which is a limitation for token masking and video generation techniques. Notably, Crossway-Diffusion [104] proposes a self-supervised loss specifically designed for CNN-based diffusion

policies [27] by redesigning the latent space of the U-net diffusion model to reconstruct the current image observation and proprioceptive features for better single task performance.

In order to predict a sequence of future actions efficiently, some recent approaches deploy transformer-based policies that encode only the current state information without any history of prior states [9, 49, 222]. Traditional token masking strategies [157] or video generation objectives [208] that rely on token sequences of multiple states for pretraining are incompatible with such single state models, since they rely on a history of prior states. MGF and CLA objectives enhance the capabilities of these single-state observation policies. MGF and CLA enable learning of versatile behavior from multimodal goals efficiently and without additional inference costs and can also be used for pretraining on action-free data.

Behavior Generation from Multimodal Goals While recent advancements in goal-conditioned robot learning have predominantly focused on language-guided methods, there is a growing interest in developing agents capable of interpreting instructions across different modalities, such as goal images, sketches, and multimodal combinations. Mutex [174] presents an imitation learning policy that understands goals in natural speech, text, videos, and goal images. Mutex further uses cross-modality pretraining to enhance the model's understanding of the different goal modalities. Steve-1 [107] is a Minecraft agent that uses a VAE encoder to translate language descriptions into the latent space of video demonstrations of the task, enabling it to follow instructions from both videos or text descriptions. Other research efforts are exploring novel conditioning methods. Examples include using the cosine distance between the current state and a goal description from fine-tuned CLIP models [171] or employing multimodal prompts [81] that combine text with image descriptions. RT-Sketch and RT-Trajectory present two new conditioning methods leveraging goal sketches of the desired scene [191] and sketched trajectories of the desired motion [56], respectively. While MDT primarily addresses the two most prevalent goal modalities, namely text and images, our framework is in theory versatile enough to incorporate other modalities like sketches.

4.3. Method

MDT is a diffusion-based transformer encoder-decoder architecture augmented with two self-supervised objectives: CLA and MGF. We first define the problem setup, then describe the diffusion formulation, model architecture, and auxiliary objectives.

4.3.1. Problem Formulation

The goal-conditioned policy $\pi_\theta(\boldsymbol{a}_{i:i+H-1} \mid \boldsymbol{s}_i, \boldsymbol{g})$ predicts an action chunk of length H, conditioned on the current observation $\boldsymbol{s}_i$ and a goal $\boldsymbol{g}$. The goal $\boldsymbol{g} \in \{\boldsymbol{o}, \boldsymbol{l}\}$ is either an image $\boldsymbol{o}$ or a language instruction $\boldsymbol{l}$. MDT learns such policies from a set of task-agnostic play trajectories $\mathcal{T}$. Each trajectory $\boldsymbol{\tau} \in \mathcal{T}$ is a sequence of state-action pairs $\boldsymbol{\tau} = \big((\boldsymbol{s}_1, \boldsymbol{a}_1), \ldots, (\boldsymbol{s}_{T_n}, \boldsymbol{a}_{T_n})\big)$, where T_n denotes the length of trajectory n. The dataset is constructed as $\mathcal{D} = \{(\boldsymbol{s}_i, \boldsymbol{a}_{i:i+H-1}) \mid (\boldsymbol{s}_i, \boldsymbol{a}_i) \in \boldsymbol{\tau}, \boldsymbol{\tau} \in \mathcal{T}\}$. During training, a goal set $\mathcal{G} = \{\boldsymbol{o}, \boldsymbol{l}\}$ is created for each sample, where $\boldsymbol{l}$ is the language annotation if available. The goal image $\boldsymbol{o}_i = \boldsymbol{s}_{i+j}$ is a future state where the offset j is sampled from the geometric distribution with bounds $j \in [20, 50]$ and probability of 0.1. MDT maximizes the log-likelihood across the play dataset,

$$\mathcal{L}_{\text{play}} = \mathbb{E}\left[\sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}} \sum_{\boldsymbol{g}\in\mathcal{G}_{\boldsymbol{s},\boldsymbol{a}}} \log \pi_\theta\left(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{g}\right)\right]. \tag{4.1}$$

4.3.2. Score-based Diffusion Policy

MDT models the goal-conditioned action distribution using the continuous-time diffusion formulation from Chapter 3. Following Karras et al. [83], a neural network D_θ approximates the score function via denoising score matching:

$$\mathcal{L}_{\text{SM}} = \mathbb{E}_{\sigma_t,\boldsymbol{a},\boldsymbol{\epsilon}}\left[\alpha(\sigma_t)\|D_\theta(\boldsymbol{a}+\boldsymbol{\epsilon},\boldsymbol{s},\boldsymbol{g},\sigma_t)-\boldsymbol{a}\|_2^2\right], \tag{4.2}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_t^2\mathbf{I})$. We use noise schedule $\sigma_t = t$ with range $[0.001, 80]$ and the DDIM sampler [184] with $N = 10$ steps. For a detailed derivation, see Section 3.3.

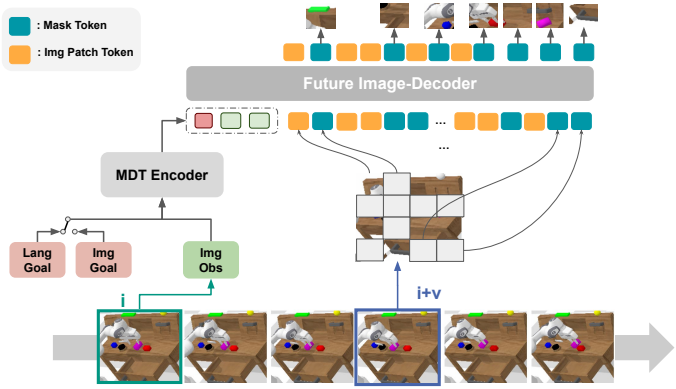


Figure 4.2.: Visualization of the Masked Generative Foresight Auxiliary Task of MDT. It starts by encoding the current observation and goal using the MDT Encoder. The resulting latent state representations then serve as conditional inputs for the Future Image-Decoder. This decoder receives encoded patches of future camera images along with mask tokens. Its task is to reconstruct the occluded patches in future frames.

The key difference from BESO is that MDT conditions on multimodal goals $g \in \{o, l\}$ rather than unimodal goal states. This requires a specialized architecture that encodes and aligns image and language goals, described next.

4.3.3. Model Architecture

MDT uses a multimodal transformer encoder-decoder architecture to approximate the conditional score function of the action sequence. The encoder first processes the tokens from the current image observations and desired multimodal goals, converting these inputs into a series of latent representation tokens. The decoder functions as a diffuser that denoises a sequence of future actions. Figure 4.1 illustrates the architecture in detail.

First, MDT encodes image observations of the current state from multiple views with image encodings. This work introduces two encoder versions of MDT: *MDT-V*, a variant with the frozen Voltron embeddings and *MDT*,

the default model with ResNets. The MDT-V encoder leverages a Perceiver-Resampler to improve computational efficiency [2]. Each image is embedded into 196 latent tokens by Voltron. The Perceiver module uses multiple transformer blocks with cross attention to compress these Voltron tokens into a set of learnable latent tokens. For MDT we use 3 for all experiments. This procedure results in a highly efficient feature extractor that capitalizes on pretrained Voltron embeddings without introducing much computation overhead. The MDT encoder uses a trainable ResNet-18 with spatial softmax pooling and group norm [27] for each camera view. Each ResNet returns a single observation token for every image. Both MDT encoder versions embed goal images and language annotations via frozen CLIP models [156] per goal-modality into a single token. After the computation of the embeddings, both MDT variants apply the same architecture comprised of several self-attention transformer layers, resulting in a set of informative latent representation tokens.

The MDT diffusion decoder denoises the action sequence with causal masking. Cross-attention in every decoder layer fuses the conditioning information from the encoder into the denoising process. The current noise level σ_t is embedded using a Sinusoidal Embedding with an additional MLP into a latent noise token. MDT conditions the denoising process to the current noise level via AdaLN-conditioning on the Transformer Decoder blocks [148]. The right part of Figure 4.1 illustrates this process, including all internal update steps. The proposed framework separates representation learning from denoising, resulting in a more computationally efficient model since the model only needs to encode the latent representation tokens once. Further, the experiments demonstrate that the proposed denoising model achieves higher performance than prior Diffusion-Transformer architectures [27]. MDT uses the same set of hyperparameters across all experiments.

4.3.4. Masked Generative Foresight

A key insight of this work is that policies require an informative latent space to understand how desired goals will change the robot's behavior in the near future. Consequently, policies that are able to follow multimodal goals have to map different goal modalities to the same desired behaviors. Whether a goal is defined through language or represented as an image, the intermediate changes in the environment are identical across these goal modalities. The

proposed *Masked Generative Foresight*, an additional self-supervised auxiliary objective, builds upon this insight.

Given the latent embedding of the MDT(-V) encoder for state $\boldsymbol{s}_i$ and goal $\boldsymbol{g}$, MGF trains a ViT to reconstruct a sequence of 2D image patches $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_U) = \texttt{patch}(\boldsymbol{s}_{i+v})$ of the future state $\boldsymbol{s}_{i+v}$, with $v = 3$ being the foresight distance used across all experiments in this work. A random subset of U of these patches is replaced by a mask token. Even though the ViT now receives both masked and non-masked patches only the reconstruction of the masked patches contributes to the loss

$$\mathcal{L}_{\text{MGF}}(\boldsymbol{s}) = \frac{1}{U} \sum_{\boldsymbol{u} \in \texttt{patch}(\boldsymbol{s}_{i+v})} \mathbf{1}_{\text{m}}(\boldsymbol{u}) \, (\boldsymbol{u} - \hat{\boldsymbol{u}})^2 , \tag{4.3}$$

where the indicator function $\mathbf{1}_{\text{m}}(\boldsymbol{u})$ is 1 if $\boldsymbol{u}$ is masked and 0 otherwise. Detailed hyperparameters for the model are summarized in Table B.2 of the Appendix.

MGF differs from existing approaches [104, 208], which require full reconstruction of images or videos. While various other masking methods exist [82], all of them aim to learn robust representations of the current state, while MGF reconstructs **future states** to include foresight into the latent embedding. MGF is conceptually simple and can be universally applied to all transformer policies. Table B.5 in the Appendix shows that MGF increases the performance of other popular imitation learning architectures like MT-ACT [9].

4.3.5. Aligning Latent Goal-Conditioned Representations

To effectively learn policies from multimodal goal specifications, MDT must align visual goals with their language counterparts. A common approach to retrieve aligned embeddings between image and language inputs is the pre-trained CLIP model, which has been trained on paired image and text samples from a substantial internet dataset [156]. However, CLIP exhibits a tendency towards static images and struggles to interpret spatial relationships and dynamics [212, 135, 129]. These limitations lead to insufficient alignment in Multi-Task Imitation Learning (MTIL) since goal specifications in robotics are inherently linked to the dynamics between the current state $\boldsymbol{s}_i$ and the desired goal $\boldsymbol{g}$. Instead of naively fine-tuning the large 300-million-parameter CLIP model, MDT introduces an auxiliary objective that aligns the MDT(-V)

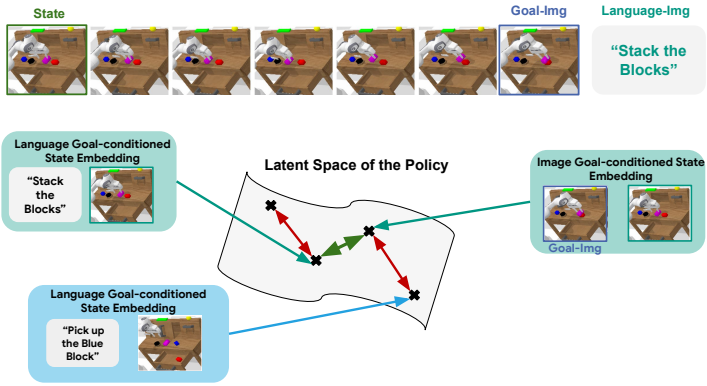


Figure 4.3.: Visualization of Contrastive Latent Alignment (CLA). During training, we enforce same observation with different goal modalities to be close in the multimodal latent space of the policy, while negative examples from other states with other goals are pushed away using InfoNCE loss objective.

state embeddings conditioned on different goal modalities. These embeddings include the goal as well as the current state information, allowing the CLA objective to consider the task dynamics.

Since CLA requires a single vector for each goal modality, the various MDT-V latent tokens are reduced via Multi-head Attention Pooling [82] and subsequently normalized. MDT uses the embedding of the static image as a representative token to compute the contrastive loss. Each training sample with multimodal goal specification $\mathcal{G} = \{\boldsymbol{o}, \boldsymbol{l}\}$ is reduced to latent vectors $\boldsymbol{z}_i^{\boldsymbol{o}}$ and $\boldsymbol{z}_i^{\boldsymbol{l}}$ for image and language goals respectively, where subscript i indexes the batch. An intuitive example of this CLA objective is visualized in Figure 4.3. CLA computes the InfoNCE loss using cosine similarity $C(\boldsymbol{z}_i^{\boldsymbol{o}}, \boldsymbol{z}_i^{\boldsymbol{l}})$

between the image-conditioned latent z_i^o and language-conditioned latent z_i^l:

$$\mathcal{L}_{\text{CLA}} = -\frac{1}{2B}\sum_{i=1}^{B}\left(\log\left(\frac{\exp\left(\frac{C(z_i^o, z_i^l)}{\upsilon}\right)}{\sum_{j=1}^{B}\exp\left(\frac{C(z_i^o, z_j^l)}{\upsilon}\right)}\right) + \log\left(\frac{\exp\left(\frac{C(z_i^o, z_i^l)}{\upsilon}\right)}{\sum_{j=1}^{B}\exp\left(\frac{C(z_j^o, z_i^l)}{\upsilon}\right)}\right)\right), \tag{4.4}$$

with temperature parameter υ and batch size B. The full MDT loss combines the Score Matching loss from Eq. (4.2), the MGF loss from Eq. (4.3), and the CLA loss from Eq. (4.4):

$$\mathcal{L}_{\text{MDT}} = \mathcal{L}_{\text{SM}} + \alpha\mathcal{L}_{\text{MGF}} + \beta\mathcal{L}_{\text{CLA}}, \tag{4.5}$$

where $\alpha = 0.1$ and $\beta = 0.1$ in most experiment settings.

4.4. Evaluation

The experiments aim to answer the following questions:

- (I) Is MDT able to learn long-horizon manipulation from play data with few language annotations?
- (IIa) Do MGF and CLA enhance the performance of MDT?
- (IIb) Does MGF improve the performance of other transformer policies?
- (III) Can MDT learn language-guided manipulation from partially labeled data in a real-world setting?

MDT is tested against several SoTA methods on both benchmarks. In addition, we evaluate MDT in a real world play setting. For all experiments we use a history length of 1 and use action chunk length of 10 for all simulation experiments and 20 for the real world experiments.

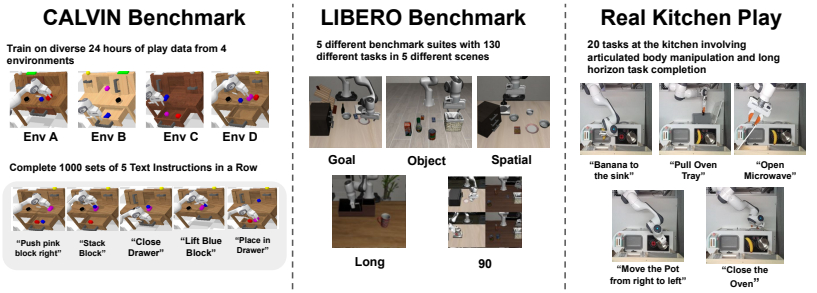


Figure 4.4.: Overview of the different environments used to test MDT. (Left) CALVIN Benchmark consisting of four environments each with unique positions and textures for slider, drawer, LED, and lightbulb. (Middle) Overview of the different tasks and scene diversity in the LIBERO benchmark, which is divided into 5 different task suites. (Right) Example tasks from the real robot experiments at a toy kitchen, where models are tested after training on partially labeled play data.

4.4.1. Evaluation on CALVIN

The CALVIN challenge [130] (detailed description in Section 2.3.5.3) consists of four environments (A, B, C, D) varying in desk shades and item layouts (Figure 4.4). We evaluate on two settings: **D→D** (6 hours of play data) to study data efficiency, and **ABCD→D** (24 hours across all environments) for multi-environment scaling. All methods are evaluated on 1000 instruction chains requiring 5 consecutive task completions, with only 1% of data annotated with language descriptions.

Baselines. We compare our proposed policy against the following state-of-the-art language-conditioned multi-task policies on CALVIN:

- **HULC:** A hierarchical play policy, that uses discrete VAE skill space with an improved low-level action policy and a transformer plan encoder to learn latent skills [129].
- **LCD:** A hierarchical diffusion policy, that extends the HULC policy by substituting the high-level planner with a U-Net Diffusion model [220] to diffuse plans.
- **Distill-D:** A language-guided Diffusion policy from [58], that extends the initial U-Net diffusion policy [27] with additional Clip Encoder for

Train	Method	No. Instructions in a Row (1000 chains)					
		1	2	3	4	5	**Avg. Len.**
D	HULC	82.5%	66.8%	52.0%	39.3%	27.5%	2.68±(0.11)
	LCD	88.7%	69.9%	54.5%	42.7%	32.2%	2.88±(0.19)
	Distill-D	86.7%	71.5%	57.0%	45.9%	35.6%	2.97±(0.04)
	MT-ACT	88.4%	72.2%	57.2%	44.9%	35.3%	2.98±(0.05)
	MDT (ours)	**93.3%**	**82.4%**	**71.5%**	**60.9%**	**51.1%**	**3.59±(0.07)**
	MDT-V (ours)	**93.7%**	**84.5%**	**74.1%**	**64.4%**	**55.6%**	**3.72±(0.05)**
ABCD	HULC	88.9%	73.3%	58.7%	47.5%	38.3%	3.06±(0.07)
	Distill-D	86.3%	72.7%	60.1%	51.2%	41.7%	3.16±(0.06)
	MT-ACT	87.1%	69.8%	53.4%	40.0%	29.3%	2.80±(0.03)
	RoboFlamingo	96.4%	89.6%	82.4%	74.0%	66.0%	4.09±(0.00)
	MDT (ours)	**97.8%**	**93.8%**	**88.8%**	**83.1%**	**77.0%**	**4.41±(0.03)**
	MDT-V (ours)	**98.6%**	**95.8%**	**91.6%**	**86.2%**	**80.1%**	**4.52±(0.02)**

Table 4.1.: Performance comparison of various policies on the CALVIN ABCD→D and D→D benchmark. The table shows the average rollout length to solve 5 instructions in a row (Avg. Len.) of 1000 chains. The numerical results are reproduced from Table I of the final RSS publication [163], including its reported ± values. MDT and MDT-V outperform the baselines included in that comparison. The source's 0.00 entry for RoboFlamingo is retained as printed; its text states that no seed standard deviation was available for this baseline.

language-goals. We use our continuous time diffusion variant instead of the discrete one for direct comparison and extend it with the same CLIP vision encoder to guarantee a fair comparison.

- **MT-ACT:** A multitask transformer policy [9, 222], that uses a VAE encoder for action sequences and also predicts action chunks instead of single actions with a transformer encoder-decoder architecture.
- **RoboFlamingo:** A finetuned vision-language model [105] containing 3 billion parameters, that has an additional recurrent policy head for action prediction. The model was pretrained on a large internet-scale set of image and text data and then finetuned for CALVIN.

We adopt the recommended hyperparameters for all baselines to guarantee a fair comparison and give an overview of our chosen hyperparameters for self-implemented baselines in Appendix B.2. Further, we directly compare the self-reported results of HULC, LCD, and RoboFlamingo on CALVIN [129, 220, 105]. The reimplemented baselines use the same language and image goal

encoders as MDT; the self-reported baselines retain their original encoders. Since RoboFlamingo only published the best seed of each model, we can not include standard deviations in their results.

Results. The results of all our experiments on CALVIN are summarized in Table 4.1. We assess the performance of MDT and MDT-V on ABCD→D and on the small subset D→D. The results are shown in Table 4.1. MDT-V sets a new record in the CALVIN ABCD→D challenge, extending the average rollout length from RoboFlamingo's 4.09 to 4.52: an increase of 0.43 completed tasks, or 10.5% relative. MDT also surpasses all other tested methods. Compared with RoboFlamingo, MDT uses fewer than 10% of the trainable parameters and requires no additional policy pretraining; the frozen CLIP and Voltron encoders retain their original pretraining. On D→D, MDT-V improves on MT-ACT, the strongest baseline in the published RSS comparison, by 0.74 completed tasks, or 24.8% relative (3.72 vs 2.98). This comparison concerns the baselines in Table 4.1; Seq-BESO is evaluated separately in Chapter 3. It also outperforms LCD [220] and the other reported methods on this smaller training set. While RoboFlamingo demonstrates commendable performance on the complete ABCD dataset, it relies on substantial training data and remains untested on the D→D subset. In contrast, MDT excels in both scenarios with remarkable efficiency.

4.4.2. Evaluation on LIBERO

Method	Lang Ann.	$\mathcal{L}_{\text{CLA}}$	$\mathcal{L}_{\text{MGF}}$	Spatial	Object	Goal	Long	90	Average
Transformer-BC [110]	100 %	×	×	71.8 ± 3.7	71.00 ± 7.9	**76.3 ± 1.3**	24.2 ± 2.6	-	-
Distill-D [58]	2%	×	×	46.8 ± 2.8	72.0 ± 6.5	63.8 ± 2.5	47.3 ± 4.1	49.9 ± 1.0	56.0 ± 3.4
MDT	2%	×	×	66.0 ± 1.9	85.2 ± 2.3	67.8 ± 4.6	65.0 ± 2.0	58.7 ± 0.8	68.5 ± 9.92
	2%	✓	×	74.3 ± 0.8	87.5 ± 2.7	71.5 ± 3.5	**65.3 ± 2.1**	66.9 ± 1.7	73.1 ± 8.81
	2%	×	✓	67.5 ± 2.1	87.5 ± 2.6	69.3 ± 2.5	63.0 ± 1.7	62.6 ± 1.0	70.0 ± 10.2
	2%	✓	✓	**78.5 ± 1.5**	**87.5 ± 0.9**	73.5 ± 2.0	64.8 ± 0.3	**67.2 ± 1.1**	**74.3 ± 9.13**

Table 4.2.: Overview of the performance of MDT and baselines with and without our proposed Self-Supervised Losses on LIBERO Task suites. All results show the average performance of all tasks averaged over 20 rollouts each and with 3 seeds. MDT outperforms Transformer-BC in several settings with only 2% of language annotations. Lang Ann. refers to Language Annotations. All numerical entries, including the reported Average and ± values, are reproduced verbatim from Table II of the final RSS publication [163].

We further evaluate models on LIBERO [110], testing their ability to learn from partially labeled data by labeling only 2% of demonstrations with task descriptions. For every task 50 demonstrations are provided but only 1 contains the respective language label. We evaluate on all five task suites (LIBERO-Spatial, LIBERO-Goal, LIBERO-Object, LIBERO-90, and LIBERO-Long) with 20 rollouts per task, averaged over 3 seeds. All policies use only static and wrist-mounted cameras. These sparse-annotation experiments differ from the fully language-labeled setting reported by Kim, Finn, and Liang [88]. Their Table I gives MDT results of 95.2%, 97.8%, and 83.0% on Spatial, Object, and Long, respectively, for training from scratch with 100% language annotations on the original unfiltered dataset. Further details for all task suites are provided in Section 2.3.5.4.

Baselines. For our experiments in LIBERO, we report the performance of MDT, Distill-D and the best transformer baseline policy from the original benchmark, which was trained with full language annotations [110].

Results. In the LIBERO task suites, summarized in Table 4.2, MDT proves to be effective with sparsely labeled data, outperforming the fully language-labeled Transformer-BC baseline on three of the four shared suites when both auxiliary objectives are used. Even without auxiliary objectives, MDT surpasses the U-Net-based Distill-D policy on every suite. Together, the auxiliary objectives increase the published average success rate across the five suites from 68.5% to 74.3%. Calculated from these reported averages, this is a gain of 5.8 percentage points, or 8.5% relative. These outcomes highlight the effectiveness of our architecture and affirmatively answer Question (I) regarding learning from sparse annotations.

4.4.3. Real Robot Experiments

We investigate research question (III) by assessing MDT's ability to learn language-guided manipulation from partially labeled data in a real-world setting.

Setup. We use the play kitchen environment described in Section 2.3.5.6 with two static RGB cameras and normalized joint space actions.

Dataset. The dataset comprises approximately 4.5 hours of teleoperated play data spanning 20 tasks. The BESO evaluation in Section 3.4.1.6 instead uses a 45-minute, 417-segment, 10-task dataset configuration; the evaluations share

Model	Avg. Single Task
MT-ACT	0.25 ± 0.43
MDT	0.51 ± 0.50
MDT + $\mathcal{L}_{\text{MGF}}$ + $\mathcal{L}_{\text{CLA}}$	**0.58 ± 0.49**

Table 4.3.: Single-task results for MDT and Baselines in the play kitchen with language goals. The average is computed over 5 rollouts for each of the 20 tasks. The poor performance of MT-ACT showcases the difficulty of this experiment. MDT performs significantly better with an additional boost through the auxiliary objectives.

the kitchen setup but use different training data configurations. Volunteers collected long-horizon demonstrations without instructions on task order, duration, or object choice. The resulting demonstrations vary from 30 to 450 seconds, containing 5–20 tasks each. We partially labeled the dataset by randomly annotating task intervals, yielding 360 labels covering approximately 20% of the data. No additional preprocessing was performed beyond trimming stationary states and cropping the teleoperator from camera views. Training a single agent on 20 skills from such unsegmented, partially labeled data presents a significant challenge. Detailed task descriptions are provided in Figure 2.10.

Single Task Evaluation. First, we test several policies to complete various single tasks from the play dataset. We test all policies on a single task setting with 5 rollouts per task. During each rollout the robot starts from a central, randomized starting position, which was not used in training. Human observers decide if a task was solved successfully for each rollout. We compare MDT with our proposed auxiliary objectives against MDT trained without them and against MT-ACT. The goals were given as language instructions. The results are averaged over 5 rollouts for each of the 20 tasks. The results are summarized in Table 4.3. The poor performance of 0.25 success rate for a strong state-of-the-art baseline such as MT-ACT highlights how challenging this setting is. In comparison, MDT achieves a respectable 0.51 success rate without and 0.58 with the auxiliary objectives. This improvement is consistent with the evaluations on the LIBERO benchmark.

Long-Horizon Multi-Task Evaluation. Finally, we test the approaches to complete several instructions in a single sequence. This requires policies to chain different behaviors together. An example instruction chain is: "Push toaster", "Pickup toast and put it to sink", "Move banana from right stove to

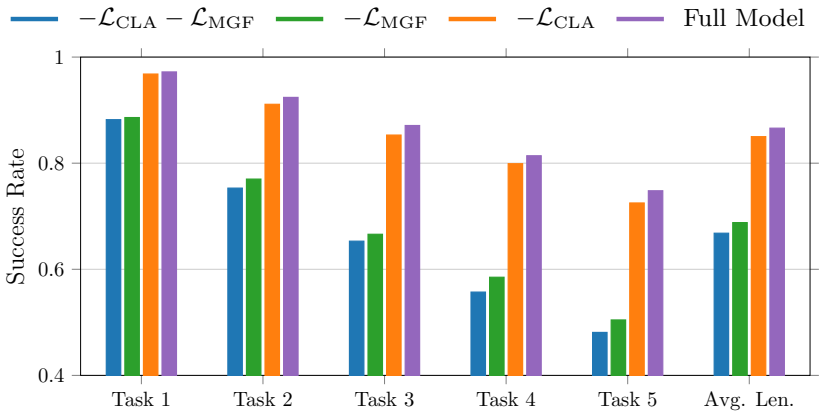


Figure 4.5.: Study on the performance of Masked Generative Foresight and the Contrastive Latent Alignment for MDT. We analyse the impact of both auxiliary tasks on the ABCD CALVIN challenge. The results show the average rollout length over 1000 instruction chains averaged over 3 seeds.

sink", "Move pot from left to right stove", "Open oven", "Open microwave". During a rollout, the model only gets the next goal, if it has completed the prior task successfully. We test all policies using 4 different instruction chains. These instruction chains are detailed in the Appendix. We evaluate language and image goal-conditioning and report the average rollout length over all chains averaged over 4 rollouts each. The results are summarized in Table 4.4. Given the increased complexity of this experiment over the single task evaluation the modest performance of all three methods is not surprising, further highlighting the difficulty of this setting. Especially when considering, that the entire dataset only encompasses $4.5h$ of unsegmented play-data. MT-ACT achieves average rollout lengths of 0.81 with language goals and 0.13 with image goals, compared with 1.38 and 0.38 for MDT without auxiliary losses. With both auxiliary losses, MDT reaches 1.56 and 0.56, respectively. Moreover, the experiment shows an even stronger contribution of the proposed auxiliary objectives to the overall performance compared to the single task evaluation. Further details are discussed in the Appendix.

The promising results of MDT compared to the state-of-the-art MT-ACT baseline in this challenging setting strongly suggests an affirmative answer

Goal Modality	Model	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Avg. Rollout Length
Language	MT-ACT	50%	18.75%	12.50%	0%	0%	0%	0.81 ± 1.01
	MDT	75%	43.75%	18.75%	0%	0%	0%	1.38 ± 1.05
	MDT + $\mathcal{L}_{\text{MGF}}$ + $\mathcal{L}_{\text{CLA}}$	81.25%	56.25%	12.50%	6.25%	0%	0%	**1.56 ± 1.06**
Images	MT-ACT	12.50%	0%	0%	0%	0%	0%	0.13 ± 0.33
	MDT	12.50%	12.50%	6.25%	6.25%	0%	0%	0.38 ± 1.05
	MDT + $\mathcal{L}_{\text{MGF}}$ + $\mathcal{L}_{\text{CLA}}$	37.50%	6.25%	6.25%	6.25%	0%	0%	**0.56 ± 1.00**

Table 4.4.: Average rollout length achieved by each method on the long-horizon real robot play kitchen setup. The performance is averaged over 4 instruction chains with 4 rollouts each. MDT clearly outperforms MT-ACT. The performance of MDT is further increased substantially by the auxiliary CLA and MGF objectives. The relatively short avg rollout lengths emphasize how challenging this setting is, even for strong state-of-the-art methods such as MT-ACT.

to Question (III), especially with the proposed auxiliary CLA and MGF objectives.

4.4.4. Evaluation of Auxiliary Losses

Next, we investigate the significance of our auxiliary self-supervised loss functions, specifically the CLA and MGF loss, on MDT's performance. Figure 4.5 shows the performance metrics of the ablated versions with and without these losses. The inclusion of MGF notably enhances MDT's performance on the CALVIN ABCD→D benchmark, improving average rollout lengths by over 25%. Detailed results supporting the essential role of these auxiliary tasks in MDT-V are presented in Table B.6 within the Appendix, showing that MDT-V surpasses all baselines with an average rollout length of 4.18 even in the absence of these two losses.

On LIBERO, using both auxiliary objectives improves success rates in four of the five suites and raises the published average by 5.8 percentage points (8.5% relative, calculated from the reported averages), as shown in Table 4.2. The results of these experiments are summarized in Table 4.2. The two losses together achieve the highest mean success rate on Spatial and LIBERO-90 and tie the highest mean on Object. Their published average of 74.3% is higher than CLA alone (73.1%) or MGF alone (70.0%), supporting the joint use of both objectives. Overall, on LIBERO the performance impact of CLA is higher than that of MGF, especially on the LIBERO-90 benchmark. A high number of different task labels, as it is the case for LIBERO-90, automatically implies a

higher number of contrastive labels for any given tasks. We hypothesize that this increased number of contrastive labels leads to the higher performance impact of CLA for LIBERO-90. The impact on the LIBERO-Goal is the lowest, given the high initial task success rate. On LIBERO-Long, the benefits are limited: CLA alone reaches 65.3% compared with 65.0% without either objective, while MGF alone reaches 63.0% and their combination reaches 64.8%. The demonstrations of the LIBERO-Long benchmark consist of several sub-tasks each with a single high-level description for the entire task. We assume that this lack of sub-goals prevents the auxiliary losses from providing notable benefits.

To investigate if MGF provides a generally beneficial auxiliary objective we integrate it with MT-ACT and evaluate the model for the full CALVIN ABCD→D benchmark, as detailed in Table B.5 in the Appendix. MGF significantly boosts MT-ACT's average CALVIN performance by 44%, without any other modifications to the model or its hyperparameters. Similarly to the MDT results, MGF also enhances the performance of MT-ACT to learn better from multimodal goals with few language annotations. These positive outcomes for MDT, along with its effective application to other transformer-based policies, positively answer research questions (IIa) and (IIb).

4.4.5. Additional Ablation Studies

This section investigates several design choices for MDT and the proposed auxiliary objectives. First, we explore the importance of using pre-trained CLIP embeddings and test MGF and CLA as pretraining objective for diffusion policies. Next, we ablate several design decisions of MGF and our proposed transformer architecture and how they impact performance.

Choice of Goal Image Encoder. We analyze the importance of using pre-trained CLIP encoders for MDT to process goal images. MDT is tested with ResNet Encoders for goal images, which are trained from scratch together with text embedding of a frozen CLIP model on CALVIN ABCD. The MDT model without our auxiliary tasks achieves an average rollout length of 3.34, which is equal to the MDT variant with the frozen CLIP embeddings. The MDT variant with both auxiliary objectives achieves an average rollout length of 4.31, which is slightly worse compared to the variant with pre-trained encoders average performance of 4.41. The detailed results of this experiment are summarized in Table B.6 of the Appendix. Overall, the ablation shows

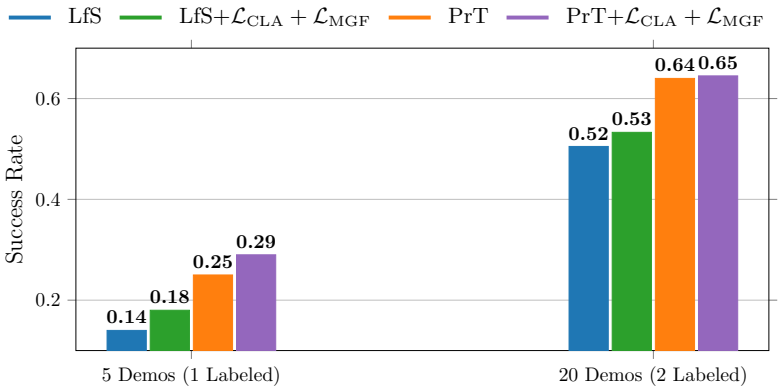


Figure 4.6.: Study on the performance of our proposed MGF and CLA objectives for pretraining on action-free data. We pretrain MDT on LIBERO-90 with the objectives and test the average performance on all LIBERO-Long tasks with different number of demonstrations. The results show the success rate averaged over 20 rollouts for all 10 tasks and 3 seeds. LfS refers to trained from scratch and PrT are all pretrained models.

that pretrained CLIP image embeddings are not required for MDT to succeed in learning from partially labeled datasets.

Pretraining with MGF and CLA. In this analysis, we assess the capability of CLA and MGF for pretraining MDT on only video-data without access to the robot actions. Therefore, we pretrain MDT with both auxiliary tasks on LIBERO-90 and test its downstream performance on LIBERO-Long, which contains unseen tasks. We compare the pretrained variant against MDT trained from scratch on this dataset and ablations, that omit auxiliary objectives for fine-tuning. For the LIBERO-Long experiment, we restrict all policies to learn from 5 or 20 demonstrations only. As specified in the published figure, the 5-demonstration setting contains one labeled demonstration (20%), and the 20-demonstration setting contains two labeled demonstrations (10%). The results of these experiments are summarized in Figure 4.6. With both auxiliary objectives used during finetuning, pretraining increases the success rate from 0.18 to 0.29 with 5 demonstrations and from 0.53 to 0.65 with 20 demonstrations, using the values displayed in the published figure. Further, these experiments demonstrate, that both auxiliary objectives also improve performance of MDT with fewer demonstrations on LIBERO-Long

independent on the pretraining. The performance increase of MDT trained from scratch with both auxiliary objectives also verifies, that CLA and MGF help MDT to learn better in scenarios with small datasets.

Masked Generative Foresight. Next, we study the different design choices of our MGF loss and compare them against ablations. Our primary focus is on assessing the impact of different masking ratios, ranging from 0.5 to 1, where 1 corresponds to a full reconstruction of the initial future image. A masking ratio of 0.75 gives the highest CALVIN mean (3.72), while 0.5 gives a slightly higher LIBERO-Spatial mean (67.8% versus 67.5%). We select 0.75, a value commonly used in other masking methods [82]. Thus, we use it as the default masking rate across all experiments in the chapter. Further details of this analysis are provided in Table B.7 in the Appendix. Additionally, we investigate the ideal foresight distance for MGF and evaluate it in two environments. MDT adopts a foresight distance of $v = 3$ as this setting consistently delivers strong performance across various scenarios. While a higher foresight distance of $v = 9$ does exhibit similar performance to a short distance of $v = 1$, it is also associated with increased variance in results. Further results of these investigations are presented in Table B.4 in the Appendix.

Transformer Architecture. MDT is tested against two Diffusion Transformer architectures previously described in [27]. The ablations are visualized in Figure B.1 of the Appendix. These comparisons are conducted on the CALVIN ABCD→D challenge, with detailed results featured in Table B.6 in the Appendix. In the first ablation study, we incorporated a noise token as an additional input to the transformer encoder. This was done to assess the effect of excluding adaLN noise conditioning. The second ablation represents the diffusion transformer architecture from [27], which does not use any encoder module. MDT-V, when trained without any auxiliary objective, achieves an average rollout length of 4.18. The ablation without adaLN conditioning only achieves an average rollout length of 3.58. Notably, the complete omission of the transformer encoder led to a significantly lower average rollout length of 1.41. The experiments show, that the additional transformer encoder is crucial for diffusion policies to succeed in learning from different goals. In addition, separating the denoising process from the encoder and using adaLN conditioning further helps to boost performance and efficiency.

4.4.6. Limitations

While MDT shows strong performance on learning from multimodal goals, it still has several limitations: 1) While we verify the effectiveness of our method in many tasks, the auxiliary objectives do not consistently improve performance on LIBERO-Long, 2) The performance impact of MGF and CLA varies across different benchmarks. On LIBERO, CLA has a higher impact, and on CALVIN, it's the opposite. 3) Diffusion Policies require multiple forward passes to generate an action sequence, resulting in lower inference speed compared to non-diffusion approaches. 4) The average rollout lengths of MDT and the baseline on the real robot multi-tasks are relatively short. We credit this to the difficulty of the setting itself. Learning from partially labeled, unsegmented, long-horizon play data is a very challenging task. We further hypothesize that the placement of the cameras is not ideal, as the robot significantly suffers from self-occlusion. The introduction on an in-hand camera could alleviate this problem.

4.5. Conclusion

In this work, we introduce MDT, a novel continuous-time diffusion policy adept at learning long-horizon manipulation from play, requiring as little as 2% language labels for effective training. To further improve effectiveness, we propose MGF and CLA as simple, yet highly effective auxiliary objectives to learn more expressive behavior from multimodal goal specifications. By reconstructing future states from multimodal goal specification and aligning these state representations in the latent space, the auxiliary objectives improve downstream policy learning without additional cost during inference. We rigorously tested MDT across 184 tasks: 34 in CALVIN, 130 in LIBERO, and 20 in the real-world kitchen. These extensive experiments not only validate our proposed auxiliary loss but also demonstrate the efficiency of the MDT policy. In the published RSS comparison, MDT-V increases average rollout length over the strongest included baseline by 10.5% relative on ABCD→D (RoboFlamingo: 4.09 to 4.52) and 24.8% relative on D→D (MT-ACT: 2.98 to 3.72). Moreover, in detailed studies, we demonstrate that our auxiliary objectives improve learning from multimodal goals.

In future work, we would like to investigate the advantages of additional goal-modalities like sketches in MDT. Furthermore, we plan to scale MDT towards a versatile foundation policy by pretraining the model on the large-scale, partially labeled OXE dataset [30].

5. Efficient Diffusion Transformer Policies with Mixture of Expert Denoisers for Multitask Learning

Chapter 4 extended BESO with self-supervised objectives for learning from partially labeled, multimodal datasets. However, scaling to larger, heterogeneous datasets from diverse embodiments requires policy backbones with sufficient capacity—yet naively scaling dense transformers incurs prohibitive training and inference costs. Unlike image generation where latency is tolerated, robot policies require real-time inference (>10 Hz) for closed-loop control.

This chapter addresses challenge **C3**: scaling diffusion policy capacity without proportionally increasing computational cost. We introduce MoDE, the first Mixture-of-Experts diffusion policy, which exploits the multitask structure of denoising—different noise levels require different computations—through noise-conditioned routing to specialized experts. Combined with expert caching at inference, MoDE achieves state-of-the-art performance across 134 tasks while using 90% fewer FLOPs than dense baselines.

The content of this chapter is based on the paper ***Efficient Diffusion Transformer Policies with Mixture of Expert Denoisers for Multitask Learning*** *(Moritz Reuss, Jyothish Pari, Pulkit Agrawal, Rudolf Lioutikov, 2025), accepted to the International Conference on Learning Representations (ICLR). Wording, notation, and formulations were revised for this thesis.*

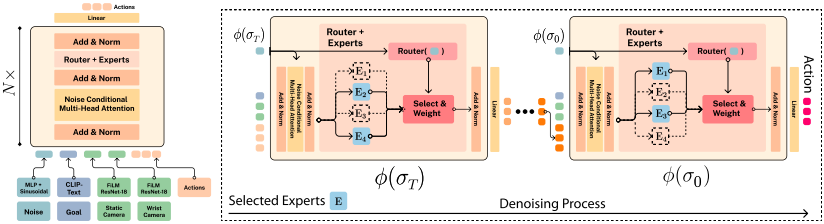


Figure 5.1.: Overview of MoDE. Left: transformer with noise-conditional self-attention and noise-conditioned routing. Right: the router activates subsets of MLP experts based on noise level during denoising.

5.1. Introduction

Sparse Mixture-of-Experts (MoE) architectures scale model capacity by activating only a subset of parameters per forward pass [177]. However, standard MoE routing based on input content does not exploit the structure of the denoising process: early steps (high noise) require coarse trajectory planning, while late steps (low noise) require fine-grained action refinement [60].

MoDE introduces *noise-conditioned routing*, which assigns tokens to specialized experts based on the current noise level rather than token content. Since the noise schedule is fixed at inference, the active experts for each denoising step are known in advance and can be cached, further reducing latency.

We evaluate MoDE on CALVIN [130] and LIBERO [110], demonstrating state-of-the-art results across 134 tasks. Comprehensive ablations analyze routing strategies, noise-injection techniques, and expert utilization. We further pretrain a 700M parameter MoDE on approximately 196,000 trajectories from a mixture based on the Open X-Embodiment dataset [30].

Our contributions:

- MoDE, a noise-conditioned MoE diffusion policy achieving 90% fewer FLOPs than dense baselines through expert caching.
- State-of-the-art results across 134 tasks, with a 57.5% mean relative improvement over the stronger of DP-T and DP-CNN in each of four benchmark settings (see Section 5.4.4).

- Comprehensive ablations on routing strategies, noise injection, and expert utilization.

5.2. Related Work

Mixture-of-Experts. MoE are a class of models where information is selectively routed through the model. The modern version of MoE was introduced in [177], where a routing or gating network conditionally chooses a subset of experts to send an input to. After Transformers [198] proved to be an effective model that scales well with data, they were modified to have expert feed-forward networks at each block of the model in [46] which presented Switch Transformers. Switch Transformers laid the groundwork that is still widely adopted in different LLM [79, 40]. This allowed for more total parameters while keeping the forward and backward FLOPs smaller than their dense counterpart, thus yielding significant performance gains. However, training both the router and experts in parallel is a non-trivial optimization problem, and it can yield suboptimal solutions such as expert collapse where experts learn similar functions instead of specializing [29]. In addition, router collapse occurs when the router selects a small subset of the experts and doesn't utilize all the experts. This is mitigated with load balancing losses [177, 46] which encourage the router to distribute inputs more evenly across experts. Multiple works have explored different methods to perform routing, such as expert choice routing [225], differential k-selection [62], frozen hashing functions [165], and linear assignment [101].

Multitask Learning in Diffusion Models. [60] demonstrated that the denoising process is inherently multi-task. Leveraging this idea, works have adopted architectures suited for multi-task learning. Some works have explicitly scheduled which parameters are specialized to which stage in the denoising process [144, 53]. In extension to this [145] uses the scheduling as guidance during training but also learns how to modulate representations based on the denoising stage. Finally, some works have employed different architectures based on the denoising stage [100].

Mixture-of-Experts in Robotics. In the context of robotics, MoE models are used in many settings without being combined with a transformer architecture. Several works use a mixture of small MLP policies, that focus on different skills in Reinforcement Learning [139, 23, 22] or for robot

motion generation[61, 96], another body of work utilizes combinations of small CNNs robot perception [164, 128]. Further applications include learning multimodal behavior using a mixture of Gaussian policies [15, 103]. Despite the extensive usage of MoE in many domains, no prior work has tried to utilize MoE together with Diffusion Policies for scalable and more efficient Diffusion Policies.

Transformers for Robot Learning. Transformer models have become the standard network architecture for many end-to-end robot learning policies in the last few years. They have been combined with different policy representations in the context of GCIL. One area of research focuses on generating sequences of actions with Variational Autoencoder (VAE) models [9, 222]. These action-chunking transformer models typically use an encoder-decoder transformer as a policy architecture. Several Diffusion Policies, such as Octo [140], BESO [162], ChainedDiffuser [209] and 3D Diffuser Actor leverage a transformer model as a policy backbone. Another direction of research treats behavior generation as discrete next-token predictions similar to auto-regressive language generation [194]. C-Bet, RT-1, and RT-2 use discretized action binning to divide seen actions into k-classes [31, 172, 18, 226], while VQ-BeT [98] learns latent actions with residual Vector Quantization. Several works have shown the advantages of using pre-trained LLM or VLM as a policy backbone, which are then finetuned for action generation [226, 56, 30, 105]. None of the recent work considers using any Mixture-of-Expert architecture for policy learning. MoDE is the first architecture to leverage MoE architecture combined with diffusion for behavior generation.

5.3. Method

In this section, we introduce MoDE, our novel MoE Diffusion Policy. First, we formulate the problem of learning a policy for GCIL. Next, we summarize the framework used in Diffusion Policies and then introduce our MoDE architecture with our novel noise-conditioned routing and noise-conditioned self-attention that enable efficient policy design. Finally, we explain our expert caching mechanism for efficient inference and explain the pretraining of MoDE.

5.3.1. Problem Formulation

Following the GCIL setup from Section 2.3.2, we learn a language-conditioned policy $\pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{l})$ from demonstrations $\mathcal{D}$. The policy predicts action chunks of length H conditioned on the current observation and language instruction. If not stated otherwise, we use $H = 10$ and no history for all experiments.

5.3.2. Diffusion Policy

MoDE employs the score-based diffusion formulation from Chapter 3, training a denoiser D_θ via:

$$\mathcal{L}_{\text{SM}} = \mathbb{E}_{t,\boldsymbol{a},\boldsymbol{\epsilon}} \left[\alpha(\sigma_t) \| D_\theta(\boldsymbol{a} + \boldsymbol{\epsilon}, \boldsymbol{s}, \boldsymbol{l}, \sigma_t) - \boldsymbol{a} \|_2^2 \right], \tag{5.1}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_t^2 \mathbf{I})$. The key innovation of MoDE lies in its MoE denoiser architecture, described next.

5.3.3. Mixture-of-Experts Denoising

We now introduce MoDE, a novel approach that employs noise-conditioned expert routing to enhance diffusion-based policies. This novel routing mechanism enables us to precompute and fuse the required experts for more efficient inference. An overview of MoDE's architecture with the routing mechanism is shown in Figure 5.1.

For language conditioning, MoDE leverages a frozen CLIP language encoder to generate a latent goal vector, while image encoding utilizes FiLM-conditioned ResNets-18/50. The model processes a sequence of input tokens $\mathbf{X} \in \mathbb{R}^{\text{tokens} \times D}$ and noise level σ_t. A linear projection layer $\phi(\sigma_t)$ encodes the noise level into a token, which is incorporated into $\mathbf{X}$. The complete MoDE architecture, $\text{MoDE}(\mathbf{X}, \phi(\sigma_t))$, consists of L transformer blocks, each specialized for different denoising phases.

We now define each block f^i as a composition of a self-attention (SA) layer and an MoE layer,

$$f^i(\mathbf{X}, \phi(\sigma_t)) = \text{MoE}(\text{SA}(\hat{\mathbf{X}}) + \mathbf{X}, \phi(\sigma_t)) + \mathbf{X}. \tag{5.2}$$

A key change in our approach is the integration of noise-aware positional embeddings. By adding $\phi(\sigma_t)$ to all tokens in $\mathbf{X}$ before self-attention:

$$\hat{\mathbf{X}} = \phi(\sigma_t) + \mathbf{X}, \tag{5.3}$$

we enable each token to adapt its attention patterns based on the current denoising phase. This design enhances denoising performance without introducing additional parameters or architectural complexity.

The self-attention mechanism follows the standard formulation [198]:

$$\mathrm{SA}(\hat{\mathbf{X}}) = \mathrm{softmax}(\frac{1}{\sqrt{D}}[\hat{\mathbf{X}}W_Q][\hat{\mathbf{X}}W_K]^T)[\hat{\mathbf{X}}W_V]. \tag{5.4}$$

Our MoE layer introduces a novel noise-conditioned routing strategy. Given N experts $\{\mathbf{E}_i\}_{i=1}^N$, the layer output is:

$$\mathrm{MoE}(\mathbf{X}, \phi(\sigma_t)) = \sum_{i=1}^{N} \mathbf{R}(\phi(\sigma_t))_i \, \mathbf{E}_i(\mathbf{X}), \tag{5.5}$$

where the routing function $\mathbf{R}(\cdot) : \mathbb{R}^D \to \mathbb{R}^N$ selects and weights experts from the noise embedding, with the same weights shared across tokens:

$$\mathbf{R}(\phi(\sigma_t)) = \mathrm{topk}(\mathrm{softmax}(\phi(\sigma_t)W_R), k) \tag{5.6}$$

Unlike traditional MoE approaches that route based on input content, MoDE's routing mechanism specifically considers the noise level. This enables specialized experts for different denoising phases, allowing for both improved performance and computational efficiency through expert caching (detailed in Section 5.3.3.1). We utilize the same method from [134] to initialize the router, which is from a truncated normal distribution with std = 0.02. We implement topk using multinomial sampling without replacement, selecting k elements according to $\mathrm{softmax}(\phi(\sigma_t)W_R)$ probabilities. To maintain gradient flow through the non-differentiable sampling process, we scale expert outputs

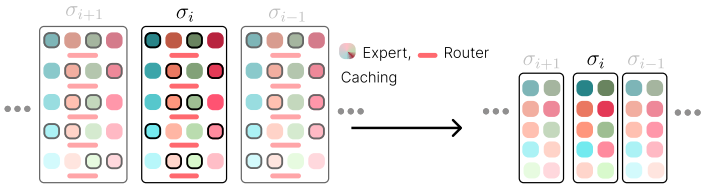


Figure 5.2.: Visualization of Router Caching. After training MoDE, the router is noise-conditioned, allowing pre-computation of the experts used at each noise level before inference. This enables removing the router and retaining only the selected experts, significantly improving network efficiency.

by routing probabilities and renormalize the selected probabilities. To prevent expert collapse, we add a weighted load-balancing loss [46] to $\mathcal{L}_{\text{SM}}$:

$$\begin{aligned} \mathcal{L}_{\text{LB}}(\mathcal{B}) &= \gamma N \sum_{n=1}^{N} f_n p_n, \\ f_n &= \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \mathbf{1}\{\mathbf{R}(\phi(\sigma_{t_i}))_n > 0\}, \\ p_n &= \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \text{softmax}(\phi(\sigma_{t_i}) W_R)_n. \end{aligned} \tag{5.7}$$

Here $\mathcal{B}$ is a minibatch, f_n is the fraction of samples routed to expert n, p_n is its mean routing probability, and $\gamma = 0.01$.

5.3.3.1. Router and Expert Caching

To make our method more efficient, we exploit the fact that our MoE is noise-conditioned, meaning where at each noise level, the routing path is deterministic and can be precomputed. By doing so, we can determine the chosen experts ahead of time for each noise level. Figure 5.2 illustrates this process. We fuse the selected expert MLPs into a single, composite MLP to reduce the computation cost. Instead of looping through each expert individually, this fused expert MLP can run in parallel, which substantially decreases the overall latency of the network. By eliminating the need for dynamically invoking each expert, we not only save time but also streamline

memory access patterns, reducing the overhead typically associated with routing decisions in a traditional MoE setup. Our caching strategy reduces the FLOPs overhead by over 80% compared to standard MoE rollouts and makes it two times faster during inference.

5.3.4. Generalist Policy Pre-Training

We pretrain MoDE on a diverse mix of multi-robot datasets curated from the OXE dataset [30]. Our training data encompasses 196k trajectories selected from six diverse datasets, featuring various robot platforms and manipulation tasks. We pretrained MoDE for 300k steps on a distributed cluster of 6 GPUs over the span of three days. For finetuning we freeze the pretrained routers in each layer and only finetune the other model components.

An Overview of the used dataset is summarized in Table 5.1. Our pretraining dataset comprises 196K trajectories from 6 different sources, with weighted sampling across BC-Z (0.259), LIBERO-10 (0.044), BRIDGE (0.188), CMU Play-Fusion (0.101), Google Fractal (0.163), and DOBB-E (0.245). This dataset includes demonstrations from diverse robot platforms including Google robots, Franka Pandas, and Hello-Stretch robots, covering a wide range of manipulation tasks.

The pretraining was conducted on 6 NVIDIA RTX A6000 GPUs with 48 GB VRAM each over 3 days, completing 300K training steps. We used a batch size of 1024 with a learning rate of $1e-4$ and weight decay of 0.1. To ensure balanced dataset mixing during training, we implemented a large shuffle buffer of 400K samples. Each dataset was individually normalized to account for different scales and ranges across the various robot platforms.

Dataset	Weight
BC-Z	0.258768
LIBERO-10	0.043649
BRIDGE	0.188043
CMU Play-Fusion	0.101486
Google Fractal	0.162878
DOBB-E	0.245176
Total	1.000000

Table 5.1.: Dataset sampling weights used for training MoDE on a small subset of trajectories. The total dataset consists of 196k trajectories.

This diverse pretraining significantly improved MoDE's zero-shot generalization, particularly on challenging

benchmarks like CALVIN ABC→D. For finetuning on LIBERO and CALVIN we freeze the routers of the model and remove the load-balancing loss and train on the local domain for 10k steps on LIBERO and 15k steps for CALVIN with a batch size of 64 per GPU and 4 GPUs. This enables very fast domain adaptation with limited training.

5.4. Evaluation

Our experiments address four questions:

(I) How does MoDE compare to prior diffusion architectures in task performance?

(II) Does large-scale pretraining improve MoDE's generalization?

(III) What are MoDE's efficiency gains over dense baselines?

(IV) How do experts specialize across noise levels, and which routing strategy works best?

We compare MoDE against prior diffusion transformer architectures [27] using similar active parameter counts. Unless a specific ablation states otherwise, the model uses 8 layers, 4 experts, and latent dimension 1024; the pretrained variant uses 12 layers. Unless stated otherwise, we use action chunk length 10, history length 1, and execute all generated actions without replanning; the state-based experiments below use chunk length 1 and history length 4. Hyperparameters are detailed in Appendix C.1.

5.4.1. State-based Experiments

We conduct initial experiments with MoDE on two established multi-task state-based environments previously described in Section 2.3.5.1 and Section 2.3.5.2: Relay Kitchen [119] and Block Push.

For Relay Kitchen, we train all models for 40k training steps with a batch size of 1024 and evaluate them 100 times as done in prior work [172, 31, 162] to guarantee fair evaluation. All reported results are averaged over 4 seeds. Training is performed on a local PC with an RTX 3070 GPU for approximately 2 hours per run. For Block Push, models are trained on the

1000 demonstrations with the same evaluation protocol, where credit is given only if blocks are pushed to the correct targets in the correct order. The policies are conditioned to push the blocks in the desired configuration using a goal state-vector. We chose an action sequence length of 1 given a history length of 4 for these experiments, which are inspired by our dense diffusion transformer baseline BESO [162] from chapter 3.

Baselines. In this setting, we compare MoDE against several SoTA goal-conditioned policies. We test two transformer architectures, C-BeT [31] and VQ-BeT [98], that predict discretized actions with an offset. C-BeT uses k-means clustering together with an offset vector while VQ-BeT leverages residual Vector Quantization to embed actions into a hierarchical latent space. Further, we test against a dense diffusion policy transformer model BESO [162]. BESO uses the same continuous-time diffusion policy combined with a dense transformer to predict a single action given a sequence of prior states. To enable a fair comparison, we chose the same hyperparameters for BESO and MoDE in both settings. The BESO and MoDE scores are the four-seed evaluations reported in the MoDE paper [160]; C-BeT and VQ-BeT follow the comparison in Lee et al. [98]. Relay Kitchen uses the requested-goal result metric. BESO's 3.73 is also reported in the sampler comparison in Table 3.5; it differs from the 3.75 result and 3.98 reward in the main ten-seed evaluation in Chapter 3.

	Block Push Success rate	Relay Kitchen Goals (out of 4)
C-BeT	0.87±(0.07)	3.09±(0.12)
VQ-BeT	0.87±(0.02)	3.78±(0.04)
BESO	0.96±(0.02)	3.73±(0.05)
MoDE	**0.97±(0.01)**	**3.79±(0.02)**

Table 5.2.: State-based goal-conditioned evaluation averaged over 4 seeds, reproduced from the MoDE study [160]. Relay Kitchen reports the mean number of requested goals completed (maximum 4); Block Push reports ordered-task success. These are separate evaluations from the 10-seed results in Chapter 3.

Results. The results of both experiments are summarized in Table 5.2. MoDE achieves a new SoTA performance on both benchmarks and outperforms the dense transformer variant of BESO in both settings. Further, MoDE achieves higher performance compared to other policy representation methods such as VQ-BeT and C-BeT.

5.4.2. Generalist Policy Evaluation

We evaluate MoDE's capabilities as a generalist policy by comparing it against two state-of-the-art models trained on substantially larger datasets from OXE: Octo (800K trajectories) [140] and OpenVLA [89](1M trajectories). We conduct this comparison using the SIMPLER benchmark, which provides real2sim variants of the BridgeV2 [200] and Google Fractal datasets used to train RT-1 [18]. The benchmark visualized in Figure 2.8 is described in detail in Section 2.3.5.5. The Google Robot evaluation uses the OXE-pretrained model followed by Fractal finetuning, as specified in Table 5.3. The WidowX evaluation tests transfer from the pretrained model. These are distinct adaptation settings, so the Google Robot results are not described as evaluation without dataset-specific finetuning.

Model	Visual Matching				Variant Aggregation				#Overall Average
	Pick Coke	Mv Near	Drawer	Avg.	Pick Coke	Mv Near	Drawer	Avg.	
RT-1-X	56.7%	31.7%	59.7%	49.4%	49.0%	32.3%	29.4%	36.9%	43.1%
RT-2-X	78.7%	77.9%	25.0%	60.5%	82.3%	79.2%	35.3%	65.6%	63.1%
Octo-Base	17.0%	4.2%	22.7%	14.6%	0.6%	3.1%	1.1%	1.6%	8.1%
OpenVLA	16.3%	46.2%	35.6%	32.7%	54.5%	47.7%	17.7%	40.0%	36.3%
MoDE (Ours)	**32.3%**	**45.4%**	**49.1%**	**42.3%**	**33.3%**	**30.0%**	**19.6%**	**27.6%**	**35.0%**

Table 5.3.: SIMPLER evaluation on Google Robot tasks. MoDE is pretrained on the OXE mixture and then finetuned on Fractal; baseline cells retain the reported comparison results. Success is full task completion. Each Avg. is the unweighted mean of the three displayed task categories, and Overall is the mean of all six displayed cells. The four-category Google Robot comparison in Chapter 6 additionally includes apple placement and uses the FLOWER evaluation setting.

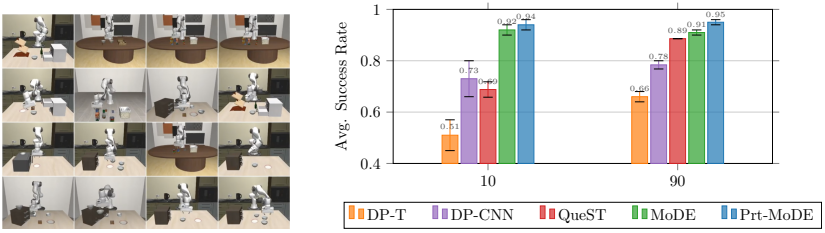
	Put Spoon on Tablecloth		Put Carrot on Plate		Stack Green on Yellow		Put Eggplant in Basket		Full Success	Mixed
Method	Grasp	Success	Grasp	Success	Grasp	Success	Grasp	Success	Average	Average
RT-1-X	16.7%	0.0%	20.8%	4.2%	8.3%	0.0%	0.0%	0.0%	1.1%	6.3%
OpenVLA	4%	0%	33%	0%	12%	0%	8%	4%	1.0%	7.63%
Octo Base	35%	12%	53%	8%	32%	0%	67%	43%	15.8%	31.25%
MoDE (ours)	29.17%	12.5%	29.17%	8.33%	8.33%	0%	37.5%	8.33%	7.3%	16.67%

Table 5.4.: SIMPLER evaluation on WidowX tasks. Grasp measures partial completion; Success measures full task completion. Full Success Average is the unweighted mean of the four Success columns. Mixed Average preserves the earlier aggregate over all eight Grasp and Success columns, as reported in the MoDE study [160]; it is not a full-task success rate.

The results of the evaluations are summarized in Table 5.3 and Table 5.4. On Google Robot tasks, MoDE achieves 35.0% overall average, with 42.3% in visual matching and 27.6% in variant aggregation after pretraining on OXE and fine-tuning on Fractal data. Notably, MoDE reaches comparable performance to the 7B parameter OpenVLA model (36.3% under the equal-weight summary used here) while requiring only 1/20th of the computational pretraining resources, demonstrating superior parameter efficiency. Furthermore, MoDE achieves this with only 700M parameters compared to 55B of RT-2-X, representing a 78x reduction in model size while maintaining competitive performance on several tasks.

On WidowX tasks, MoDE achieves 7.3% mean full-task success, compared with OpenVLA's 1.0% and Octo-Base's 15.8%. The earlier 16.7%, 7.63%, and 31.25% figures average partial grasp and full-task scores and are retained as Mixed Average in Table 5.4. Octo-Base therefore outperforms MoDE on this setting. RT-2-X is not evaluated in this WidowX table; its Google Robot score cannot serve as a WidowX baseline. These results show both the potential and the remaining transfer limitations of the smaller policy.

5.4.3. Long-Horizon Multi-Task Experiments



(a) LIBERO-90 Tasks

(b) Results for LIBERO-10 and LIBERO-90

Figure 5.3.: Visualization and Results for LIBERO environment. (a) Example tasks from the LIBERO benchmark of Liu et al. [110]. (b) Our evaluation results on LIBERO-10 and LIBERO-90, averaged over 3 seeds with 20 rollouts for each task.

Next, we first evaluate MoDE on the LONG-challenge and LIBERO-90 challenge of the LIBERO benchmark [110]. Figure 5.3a visualizes a few examples of these tasks (see Section 2.3.5.4 for detailed description). All environments feature two cameras: a static one and a wrist-mounted camera, used to encode

the current observation using FiLM-ResNets-18/50. We test each policy 20 times on each task and report the average results over 3 seeds. MoDE and all other diffusion architectures use FiLM-conditioned ResNets-18/50 with a CLIP sentence embedding to encode the goal and the images.

Baselines. We compare MoDE against three state-of-the-art baselines: 1) The Diffusion Transformer (DP-T) architecture [27], which conditions on noise and observations using a cross-attention module. 2) The standard Diffusion Policy CNN-based architecture (DP-CNN). 3) QueST [131], a transformer-based policy that learns discrete action representations using vector-quantized embeddings of action sequences. We tested all baselines ourselves, except for QueST, whose results are taken directly from their paper.

Results. The performance of all models on the benchmark is summarized in Figure 5.3b. Overall, MoDE achieves the highest average performance in both benchmarks, while the QueST baseline is the second best in the LIBERO-90 setting and the CNN-architecture is second best in the long horizon setting. These results demonstrate MoDE's ability to learn long-horizon tasks with high accuracy. The performance gap is more pronounced in the challenging LIBERO-10 experiment, where MoDE is the first policy to achieve an over 90% success rate. Against DP-CNN, the stronger diffusion baseline in these two settings, the relative gains are 16.7% on LIBERO-90 (0.91 versus 0.78) and 26.0% on LIBERO-10 (0.92 versus 0.73), or 21.3% averaged across the two settings. The four-setting mean also includes the two CALVIN challenges and is defined below. The pretrained MoDE variant achieves an even higher performance in both settings, showing the potential of diverse pretraining. This showcases MoDE's ability to achieve state-of-the-art performance with a more efficient use of computational resources.

5.4.4. Scaling Multi-Task Experiments

Scope of the aggregate improvement. The headline gain compares the non-pretrained model with the stronger of DP-T and DP-CNN separately in CALVIN ABC→D, CALVIN ABCD→D, LIBERO-90, and LIBERO-10. The relative gains are 151.1% (3.39/1.35−1), 36.1% (4.30/3.16−1), 16.7% (0.91/0.78−1), and 26.0% (0.92/0.73 − 1); their unweighted mean is 57.5%. Thus, 57.5% describes four settings, whereas 21.3% describes the two LIBERO settings only.

Train→Test	Method	Active Params in Million	PrT	No. Instructions in a Row (1000 chains)					
				1	2	3	4	5	**Avg. Len.**
ABCD→D	DP-CNN	321	×	86.3%	72.7%	60.1%	51.2%	41.7%	3.16±0.06
	DP-T	194	×	78.3%	53.9%	33.8%	20.4%	11.3%	1.98±0.09
	Seq-BESO	50	×	93.6%	87.3%	81.0%	74.9%	67.7%	4.05±0.04
	RoboFlamingo	1000	✓	96.4%	89.6%	82.4%	74.0%	66.0%	4.09±0.00
	GR-1	130	✓	94.9%	89.6%	84.4%	78.9%	73.1%	4.21±0.00
	MoDE	277	×	96.6%	90.6%	86.6%	80.9%	75.5%	4.30±0.02
	MoDE (Prt)	436	✓	**97.1%**	**92.5%**	**87.9%**	**83.5%**	**77.9%**	**4.39±0.04**
ABC→D	DP-CNN	321	×	63.5%	35.3%	19.4%	10.7%	6.4%	1.35±0.05
	DP-T	194	×	62.2%	30.9%	13.2%	5.0%	1.6%	1.13±0.02
	Seq-BESO	50	×	76.9%	53.3%	35.8%	24.2%	16.1%	2.06±0.13
	RoboFlamingo	1000	✓	82.4%	61.9%	46.6%	33.1%	23.5%	2.47±0.00
	SuSIE	860+	✓	87.0%	69.0%	49.0%	38.0%	26.0%	2.69±0.00
	GR-1	130	✓	85.4%	71.2%	59.6%	49.7%	40.1%	3.06±0.00
	MoDE	307	×	91.5%	79.2%	67.3%	55.8%	45.3%	3.39±0.03
	MoDE (Prt)	436	✓	**96.2%**	**88.9%**	**81.1%**	**71.8%**	**63.5%**	**4.01±0.04**

Table 5.5.: Performance comparison on the two CALVIN challenges. The table reports average success rates for individual tasks within instruction chains and the average rollout length (Avg. Len.) to complete 5 consecutive instructions, based on 1000 chains. Zero standard deviation indicates methods without reported standard deviations. "Prt" denotes models requiring policy pretraining. Parameter counts exclude language encoders.

We evaluate MoDE on the **CALVIN Language-Skills Benchmark** [130], a comprehensive image-based benchmark for GCIL. In the **ABCD→D** challenge, models learn from 22, 966 interaction sequences across four environments (A, B, C, D), each comprising 64 timesteps and 34 diverse tasks requiring complex skill chaining (Figure 2.6a). Following the official protocol, we test all policies on 1000 instruction chains of 5 sequential tasks in environment D (Figure 2.6b). Models receive 1 point per completed task and must complete each task to progress. We report average sequence length over 3 seeds with 1000 instruction chains each.

Baselines. We test MoDE against several methods specialized for learning language-conditioned behavior and against other baseline diffusion policy architectures. We include Seq-BESO [162], the score-based diffusion policy introduced in Chapter 3, as a baseline to assess improvements from our mixture-of-experts architecture. We also compare MoDE against RoboFlamingo and GR-1. RoboFlamingo is a fine-tuned Vision-Language-Action model, that contains around 3 billion parameters and has been pre-trained on diverse internet data. GR-1 is a generative decoder-only Transformer pretrained on large-scale video generation and then co-finetuned on CALVIN [208]. If

available, we report the average performance of all prior work directly from their paper, given the standard evaluation protocol in CALVIN [129].

Results. MoDE achieves the highest average success rate across all policies, even without pretraining (Table 5.5). This surpasses extensively pretrained baselines like RoboFlamingo and GR-1, as well as Seq-BESO by a significant margin (4.39 vs 4.05 on ABCD→D, 4.01 vs 2.06 on ABC→D). The improvement over Seq-BESO demonstrates the effectiveness of our mixture-of-experts architecture combined with diverse pretraining and noise-conditioned routing for multi-task learning. While GR-1 uses fewer parameters (130M vs 436M), MoDE requires significantly fewer FLOPs during inference (1.53 vs 27.5 GFLOPS) while maintaining comparable speed (12.2 vs 12.6 ms). The combination of state-of-the-art performance and computational efficiency makes MoDE particularly suitable for multitask applications.

5.4.5. Generalization Experiments

Finally, we then extend our investigation to the **ABC→D** challenge in the second phase, testing MoDE's zero-shot generalization abilities. In this experiment, models are only trained on data from the first three CALVIN environments A,B,C and tested on the unseen setting of environment D, which has different positions of relevant objects and texture of the table. This requires policies, that are able to generalize their learned behavior to new environment configurations and different textures, which is especially challenging. We evaluate MoDE trained from scratch and MoDE pretrained on a sub-set of Open-X-Embodiment Data. This enables us to study the zero-shot performance and pretraining effectiveness of MoDE.

Baselines. For this experiment, we compare MoDE against the previous CALVIN baselines, with the addition of SuSIE [12], a hierarchical policy utilizing a finetuned image-generation model, InstructPix2Pix [19], to generate goal images, which guide a low-level diffusion policy. The high-level goal generation model requires large-scale pretraining. SuSIE's results are based on 100 rollouts only, without standard deviation, due to the computational cost of generating subgoal images.

Results. The results of this experiment are summarized in the lower part of Table 5.5. MoDE outperforms all tested baselines and surpasses all other Diffusion Policy architectures by a wide margin. Further, by pretraining

MoDE on diverse robotics data, we achieve a new SoTA performance of 4.01. Mixture-of-Experts models enhance scaling performance but also demonstrate strong zero-shot generalization capabilities (Question (I)). In addition, we can answer Question (II) by concluding that pretraining boosts performance in challenging zero-shot settings.

5.4.6. Computational Efficiency of MoDE

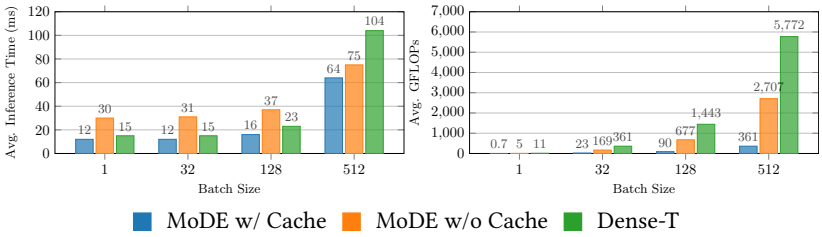


Figure 5.4.: Computational efficiency comparison between MoDE and a Dense-Transformer model with the same number of parameters. Left: Average inference speed over 100 forward passes for various batch sizes. Right: FLOPs count for MoDE with router cache and without compared against a dense baseline. MoDE demonstrates superior efficiency with lower FLOPs count and faster inference thanks to its router caching and sparse expert design.

We compare MoDE against a dense transformer baseline with similar parameters by measuring average inference time and FLOPs across batch sizes. As shown in Figure 5.4, MoDE with caching significantly improves computational efficiency - at batch size 1, inference is 20% faster (12ms vs 15ms), while at batch size 512, MoDE requires 16x fewer FLOPs (361 vs 5,772 GFLOPS) and achieves nearly 40% faster inference (64ms vs 104ms). These results demonstrate that MoDE delivers both superior task performance and substantial computational efficiency through its architecture and caching mechanism.

5.4.7. Ablation Studies

We conduct ablations to understand MoDE's design choices, addressing questions (III) and (IV): computational efficiency, routing strategies, and expert

utilization. Only the CALVIN ABC ablation runs use a separate, weaker training run than the main ABC benchmark. We assess their relative performance against the corresponding ablation baseline.

5.4.7.1. Design Decisions

We ablate noise conditioning and MoE strategies on LIBERO-10 (Table 5.6).

Noise-Injection Ablations. Our experiments reveal the importance of proper noise conditioning in MoDE. The full MoDE model, which uses both input noise tokens and noise-conditioned self-attention, achieves the best performance with an average success rate of 0.92. Removing the input noise token slightly decreases performance to 0.90, highlighting the complementary nature of both conditioning methods. Using only the noise token for conditioning, without noise-conditioned self-attention, further reduces performance to 0.85. Interestingly, using FiLM conditioning [149], a common approach in image-diffusion [148], yields the lowest performance in this group at 0.81. These results underscore the effectiveness of our proposed noise conditioning strategy in MoDE, demonstrating a clear performance advantage of 0.08 over the FiLM approach.

MoE Ablations. Next, we ablate several design decisions regarding Mixture-of-Experts. First, we test the topk number of used experts. Setting topk to 1 only marginally lowers the average performance from 0.92 to 0.91. MoDE maintains robust performance even with a single expert. We also examine the effect of using a shared expert, where the model consistently employs the same expert in all cases. This approach achieves a comparable average success rate of 0.90. Different choices for the token-distribution loss are also tested. While MoDE uses $\gamma = 0.01$ as a default value, we experiment with γ values of 0.1 and 0.001, which result in average success

	Avg. Success.
MoDE	**0.92**
- Input Noise Token	0.90
- Noise-cond Attention	0.85
FiLM Noise Conditioning	0.81
TopK=1	0.91
Shared Expert	0.90
$\gamma = 0.1$	0.90
$\gamma = 0.001$	0.86
256 Embed Dim	0.86
512 Embed Dim	0.87

Table 5.6.: Ablation Studies for MoDE on LIBERO-10. All results are averaged over 3 seeds with 20 rollouts each.

Model	Block Push	Relay Kitchen	CAL ABC	CAL ABCD	L-10	Average
Dense T	0.96±0.02	3.73±0.12	**2.83±0.19**	4.13±0.11	0.91±0.02	0.839±0.144
Token-Router	0.97±0.01	**3.85±0.03**	2.67±0.04	4.29±0.08	0.90±0.01	0.845±0.161
σ_t-Router	*0.97±0.01*	3.79±0.04	*2.79±0.16*	**4.30±0.02**	**0.92±0.02**	**0.851±0.151**

Table 5.7.: Overview of the performance of all different token routing strategies used for MoDE across 5 benchmarks. We mark the best result for each environment in **bold** and the second best in *italics*. We use CAL to represent CALVIN. To average the results, we normalize all scores and compute the average over all environments. Only the CALVIN ABC ablation results use a separate, weaker training run than the main ABC benchmark.

rates of 0.90 and 0.86, respectively. These results indicate that a γ value of 0.01 strikes the best performance.

Latent Dimension. We investigate the impact of varying the latent dimension in MoDE, testing dimensions of 256, 512, and 1024 (our default). The results show that increasing the latent dimension from 256 to 512 yields a modest performance improvement from 0.86 to 0.87, while further increasing to 1024 provides a more substantial boost to 0.92. This suggests that a larger latent dimension allows MoDE to capture more complex representations, leading to improved performance.

5.4.7.2. Optimal Routing Strategy for Diffusion Transformers

Next, we answer Question (IV) by testing different routing strategies for our diffusion-transformer policy across several environments. We compare two routing strategies: (1) token-conditioned routing (standard in LLMs) and (2) noise-conditioned routing. We test these strategies in five experiments and report the average performance over 3 seeds: Noise-only Routing achieves an average normalized performance of 0.851, slightly outperforming Token-only Routing, which achieves 0.845. Detailed results are summarized in Table 5.7. The CALVIN ABC runs in the routing ablation reported in Table 7 of the MoDE paper [160] and in the top-k study in Appendix C.3 use a separate, weaker training run than the main ABC evaluation in Table 5.5. Their 2.79 and 3.34 scores support relative comparisons within the respective ablations; the main non-pretrained ABC benchmark reports 3.39. This distinction concerns the ABC ablations only. The results demonstrate the effectiveness of our

proposed routing strategy. While the performance difference is small, Noise-only Routing offers an additional advantage: the ability to predict all used experts based on noise levels once before roll-outs, enabling faster inference, as described in Section 5.3.3.1. This is particularly beneficial for robotics applications.

5.4.7.3. How does the model distribute the tokens to different experts?

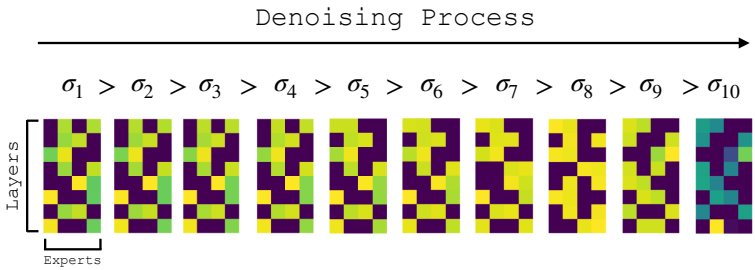


Figure 5.5.: Visualized Expert Utilization. The average usage of all experts in MoDE across all layers is shown. Purple color corresponds to low usage and yellow color to high one, and each image is separately normalized. The average activation shows that MoDE learns to utilize different experts for different noise levels.

To address Question IV, we analyzed how MoDE distributes tokens to different experts using a pre-trained model. Figure 5.5 visualizes the average usage of each expert in each model layer during inference across various noise levels, using 10 denoising steps for clarity. Our analysis reveals that MoDE learns to utilize different experts for various noise levels, suggesting that the router has specialized for different noise regimes. A transition in expert utilization occurs around σ_8. In the first layer, the model learns an expert specialized for low-noise levels, primarily used in the last denoising step at σ_{min}. We conduct more ablations studies with our pretrained model and various other versions of MoDE in Appendix C.3 of the Appendix. These findings confirm that MoDE learns meaningful expert specialization based on noise levels.

5.4.7.4. Scaling with More Experts

We evaluate 2, 4, 6, and 8 experts on CALVIN ABCD and ABC (Figure C.1). Performance peaks at 4 experts; adding more degrades results, possibly due to overfitting or routing complexity. A dense baseline with matched latent dimension (Dense-small) consistently underperforms, confirming MoE's parameter efficiency.

We hypothesize that 4 experts represents an optimal trade-off for noise-only routing: the specialization patterns in Figure 5.5 show experts specialize by noise region, and finer partitioning provides no additional benefit. This contrasts with token-routing MoE, which scales to hundreds of experts [90].

5.4.7.5. Flow Matching Finetuning for MoDE

We investigate the compatibility of MoDE with alternative denoising objectives by combining our architecture with flow matching [3, 109] from Section 2.1.4. Using our OXE-pretrained MoDE model finetuned with an fm objective, we evaluate performance with reduced inference steps on the ABC→D setting.

Method	Steps	No. Instructions in a Row (1000 chains)					
		1	2	3	4	5	**Avg. Len.**
MoDE	10	96.2%	88.9%	81.1%	71.8%	63.5%	4.01±0.04
MoDE-FM	2	93.5%	86.2%	77.5%	68.8%	60.5%	3.87±0.06

Table 5.8.: Performance comparison of MoDE with standard vs Flow Matching objective on CALVIN ABC→D. MoDE-FM achieves comparable performance with just 2 denoising steps.

As shown in Table 5.8, MoDE-FM maintains strong performance while requiring only 2 denoising steps compared to the original 10 steps. The modest performance drop (4.01 vs 3.87 average rollout length) suggests that our noise-specialized experts successfully adapt to different denoising objectives. Future work could explore combining MoDE with Flow Matching to further reduce inference steps while maintaining high performance.

5.4.8. Limitations

MoDE still has certain limitations:

- In our experiments, we find that MoDE exhibits a slightly higher standard deviation compared to the baselines. We hypothesize that the router's initialization has significant impact on overall optimization, requiring future work on stabilizing routing models.
- when visualizing expert utilization, in some of our experiments we noticed that only a subset of the total experts were being utilized - a phenomenon known as expert collapse [29].
- Faster Inference is dependent on Expert Caching. Given the modest size of the latent dimensions of MoDE naively using MoE design does not result in faster inference.
- MoDE using noise-only routing does not scale towards more than 4 experts. This limits efficiency gains compared to MoE using token-routing that can be scaled towards several hundred experts [90].

5.5. Conclusion

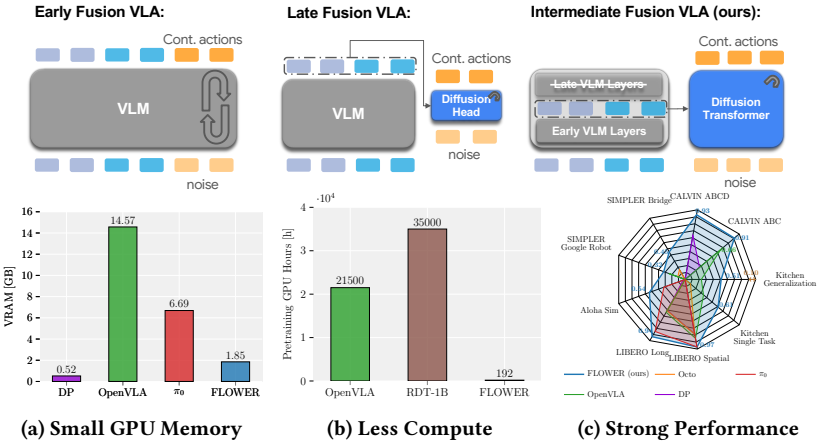
In this work, we introduced Mixture-of-Denoising Experts (MoDE), a novel Diffusion Policy that leverages a mixture of experts Transformer to enhance the performance and efficiency of diffusion policies. We also proposed a noise-conditioned routing strategy for learning specialized experts within our model. In our extensive experiments and ablation studies across diverse benchmarks, we demonstrated the advantages of MoDE to outperform prior Diffusion Policies with a lower number of parameters and 90% less FLOPS during inference. In future work, we want to experiment with more routing strategies, such as expert-choice routing [225].

6. FLOWER: Democratizing Generalist Robot Policies with Efficient Vision-Language-Flow Models

In the preceding chapters, we introduced several contributions towards scaling BC to versatile agents. Chapter 3 established score-based diffusion policies as the policy representation for learning from play data (**C1**). Next, MDT addressed the challenges of training on partially labeled datasets (**C2**) using self-supervised learning with CLA and MGF. Chapter 5, introduced MoDE, a noise-routing MoE-based diffusion policy, which was pretrained on robot-agnostic robot play with efficient inference and improved generalization (**C3**).

However, all prior policies rely on separate, pretrained encoders for image and text representations. Their vision and language alignment is only trained during the imitation learning phase. They do not leverage the semantic grounding and cross-modal alignment achieved by generalist VLMs. In addition, our prior pretraining paradigms of MoDE were limited to single action spaces Delta-End-Effector.

We introduce FLOWER, a novel compact, generalist VLA policy to address these limitations through a unified, efficient VLA architecture. Using our contributions of intermediate modality fusion for bridging the gap between VLM and action generation efficiently (**C3**) and Action-specific global AdaLN to tackle heterogeneous training on cross-robot and action spaces (**C2**). These ideas are combined with fast and expressive Rectified Flow (RF) action generation (**C1**) towards a unified and general VLA. We systematically investigate key design decisions for building compact VLAs. This final architecture,



(a) Small GPU Memory **(b) Less Compute** **(c) Strong Performance**

Figure 6.1.: Intermediate fusion for efficient VLA policies. Our fusion strategy (top-right) strategically prunes VLM layers while enhancing Flow Transformer capacity in parameter-constrained settings. This approach informs FLOWER, a novel, 950*M* VLA that achieves competitive performance across 10 benchmarks using only 1% of the pretraining compute of models like OpenVLA [89], while maintaining a small memory footprint across diverse embodiments and action spaces (bottom).

FLOWER, achieves state-of-the-art performance across 10 benchmarks. Crucially, by requiring only 950M parameters and 1% of the pretraining compute of prior models, FLOWER demonstrates the potential of compact, efficient VLAs to democratize generalist robotics research.

*The content of this chapter is based on the paper **FLOWER: Democratizing Generalist Robot Policies with Efficient Vision-Language-Flow Models** (Moritz Reuss, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Otto, Rudolf Lioutikov, 2025), published in the Conference on Robot Learning (CoRL). Wording, notation, and formulations were revised for this thesis.*

6.1. Introduction

Achieving versatile robot policies that generalize across tasks and embodiments requires strong semantic grounding from vision and language. Prior

multimodal policies like MDT and MoDE use frozen CLIP embeddings, which provide natural language understanding but lack the deep cross-modal alignment needed for robust instruction following in novel scenarios. VLAs address this by leveraging pretrained VLMs, which learn richer vision-language alignment from internet-scale data [11, 89, 39]. However, current VLAs contain billions of parameters [89, 113], limiting accessibility for research and deployment. Designing compact alternatives reveals a parameter budget trade-off (Figure 6.1): a deep VLM backbone leaves insufficient capacity for the action head to model multimodal trajectories, while a shallow VLM sacrifices the semantic features needed for instruction following. Further, conditioning the action head on full VLM forward passes slows training convergence and increases inference latency.

To resolve this budget–tradeoff, we introduce four key contributions. First, we propose *intermediate-level fusion*: we prune between 30% and 50% of the pretrained VLM's layers and condition our Flow Transformer on the resulting intermediate embeddings, thereby retaining semantic grounding while reclaiming parameters. Second, we develop a *global action-space AdaLN* mechanism: an action-specific LayerNorm controller within the diffusion transformer that reduces head parameters by 20% without any loss of accuracy or expressivity. Third, we provide extensive ablations across multiple benchmarks that evaluate design choices and yield additional insights into which VLM architectures and pretraining objectives are best suited for efficient VLA adoption.

Leveraging these contributions and insights, we present our fourth contribution: **FLOWER**, a 950M parameter VLA policy that converges quickly, cuts pretraining cost on heterogeneous robotics data by 99%, and yields competitive performance with current VLAs across 190 tasks in 10 benchmarks across simulation and real world settings.

6.2. Related Work

Early imitation-learning studies demonstrated that policy performance scales with dataset size. For example, Pinto and Gupta [152] used over 50K grasps and observed steady improvements in manipulation proficiency. Building on this, the OXE benchmark provided 1.4M trajectories across more than

20 robot embodiments, enabling research into generalist policies [30]. Several diffusion-policy approaches directly train on OXE without leveraging pretrained VLMs. Octo [140] applies a transformer-based diffusion policy to delta end-effector actions but lacks a pretrained vision–language encoder and model capacity, which limits its generalization as observed on benchmarks such as SIMPLER [106]. RDT introduces a 1.2B-parameter diffusion transformer paired with an 11.4B-parameter pretrained language encoder model and ViT. However, its one-month pretraining on 48 A100 GPUs highlights the need for more computationally efficient VLA pretraining [113]. Addressing this need, our contributions yield versatile VLAs in only 200 GPU hours of pretraining.

To address generalization, recent methods incorporate large pretrained VLMs into the policy. OpenVLA fine-tunes a 7.7B-parameter VLM for discrete end-effector action prediction, yet its size makes real-robot deployment challenging [89]. Several other VLAs apply discrete action prediction [18, 226, 39, 16, 173, 43]. RoboDual improves upon OpenVLA, by integrating a small diffusion transformer with OpenVLA via asynchronous updates, while Latent Bridge uses a fine-tuned VLM to generate latent commands that condition a diffusion policy [20, 178]. Similar to FLOWER, π_0 [11] and GR00T-N1 [10], also introduce generalist flow-based VLA with more than 2B parameters, which are both trained on a closed-source cross-embodiment datasets. Despite their effectiveness, these approaches retain very large VLMs with high-memory requirements and slow convergence. In contrast, FLOWER matches these methods with less than 1 billion parameters and significantly lower memory and fully open source training.

Cross-Embodiment Learning A core challenge in robotics is learning a unified policy for heterogeneous embodiments with distinct action and sensor spaces. Early approaches often applied modular policies [70] or hardware-conditioned representations [26], and some leveraged graph-based representations to generalize across different robot hands [146, 214]. However, these efforts tended to focus on smaller datasets or simplified environments. Recent work on large-scale cross-embodiment includes RoboCat [16] and PolyBoT [214], which use action tokenization or hierarchical controllers, respectively. Liu et al. [113] propose a unified 258-dimensional action space (RDT-1B) with fixed action prediction length of 64 for all action spaces, while CrossFormer [36] employs separate action heads with a continuous action prediction head for different embodiments but lacks a pretrained vision-language component for generalization to diverse instructions. By incorporating an

action-type Global AdaLN conditioned Flow Transformer with a pretrained VLM, FLOWER efficiently handles multiple embodiments while maintaining both action expressiveness and semantic understanding.

Rectified Flow and Diffusion Models in Robotics. Diffusion models [66, 188, 184] have become widely used for generating continuous robot actions from visual inputs [27, 140, 162, 84], offering multi-modal behavior [78] and good scaling with large datasets [140, 113, 160]. More recently, *Rectified Flow* [42, 3, 109] has emerged as a promising alternative, enabling a straight-line probability path for action sampling that requires few discretization steps. In robotic policy learning, ActionFlow [50], π_0 [11] and others [221, 17] showed rectified flow can generate actions more rapidly than standard diffusion. Such fast inference is crucial for high-frequency robot setups like Aloha [49]. Yet, these works are typically confined to a single embodiment or a single action space. By contrast, FLOWER is the first completely open-source policy to apply rectified flow as a *generalist* policy component, unifying expressive and multimodal flow-based action generation with diverse vision-language contexts. While π_0 published their weights for finetuning, the code for pretraining and their dataset remains closed source.

Efforts to reduce model size and finetuning overhead include using smaller VLM backbones or alternative fusion strategies. TinyVLA attaches a compact diffusion head to a lightweight VLM with a late-fusion approach and no pretraining [206]. Fusion strategies range from early fusion, that merges raw VLM outputs with the action generator at the input stage, to late fusion, where modalities are processed separately and combined only after independent streams [206, 113, 69, 102]. In contrast, intermediate fusion (our contribution) injects mid-level VLM tokens into the Flow Transformer, allowing selective pruning of the VLM while preserving semantic richness. While DeeR-VLA [216] has explored early exit strategies for continuous action prediction VLAs, FLOWER significantly surpasses it across benchmarks by wide margins thanks to our novel flow-based fusion design. Beyond model architecture, the choice of data sources and automated data collection are key to pretraining robust policies. Concurrent work has also explored intermediate-fusion [10] without any ablation and insights on the impact. FLOWER advances efficient VLAs using intermediate-modality fusion and global-adaln conditioning, enabling faster training and inference while achieving SoTA performance on diverse benchmarks.

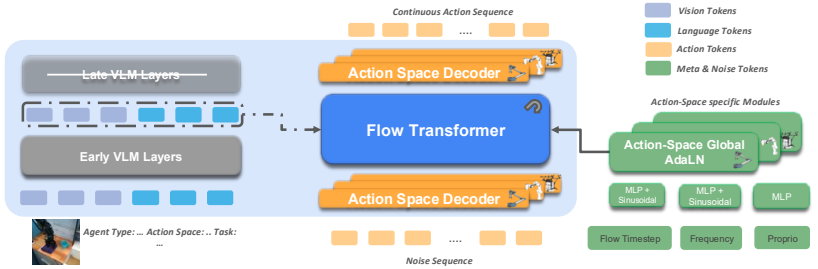


Figure 6.2.: Visualization of the FLOWER architecture. A fine-tuned VLM processes multimodal inputs and integrates intermediate features into a Flow Transformer via cross-attention. The model predicts velocity fields with Global AdaLN-Zero conditioning for the action space, embodiment, and temporal metadata.

6.3. Method

We learn an efficient, generalist policy π_θ that generates actions conditioned on state $\boldsymbol{s}$, text goal tokens $\boldsymbol{g}$, and meta-embodiment information $\boldsymbol{e}$. This task is challenging due to heterogeneity in action spaces (varying degrees-of-freedom, control modes), observation spaces (different sensors), and task specifications (diverse language goals). Given trajectories from multiple embodiments, we learn a unified policy that generalizes across robots, tasks, and observation formats by maximizing $\mathcal{L}_{\text{IL}} = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},\boldsymbol{g},\boldsymbol{e})\sim\mathcal{D}}[\log \pi_\theta(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{g}, \boldsymbol{e})]$. FLOWER does not use history of frames and predicts a chunk length of $H = 10$ if not noted otherwise.

6.3.1. Intermediate Modality Fusion Vision-Language-Action-Models

On a high level, flow-based VLA consists of two main components: A VLM pretrained on internet-scale vision and text data to encode state information and text and a flow prediction module to generate a sequence of actions conditioned on the current context from Gaussian noise.

We propose to fuse pretrained vision–language representations with Flow Transformer embeddings at an intermediate stage to balance semantic depth

and computational efficiency. We hypothesize that intermediate representations retain useful visual and language context while avoiding the cost of the full generative backbone. We therefore extract hidden states from an intermediate stage of the VLM after jointly encoding visual (ViT) and textual tokens. This is an architectural hypothesis tested by the fusion and pruning ablations in Section 6.4.1, rather than a general claim about the semantics of a particular quarter of transformer layers. We prune the VLM based on its architecture: For **Encoder–Decoder VLMs** (Florence-2 [211]), we remove the full decoder, keeping only the encoder LLM layers. This reduces the number of layers by 50% while increasing its performance and efficiency. For **Decoder-Only VLMs** (SmolVLM2-Video [126]), we drop the final 30% of transformer layers. This targeted pruning cuts 20–35% of parameters and reduces per-step latency. In Section 6.4.1, we provide ablations to verify our intermediate fusion design. The full architecture is visualized in Figure 6.2.

Next, we take the projected VLM latent tokens and inject them into the Flow Transformer via cross-attention. For increased stability, we map the VLM hidden states through a linear layer followed by RMSNorm [219]. This design conditions each Flow layer on semantically rich VLM features, preserving spatial and contextual structure and enabling faster policy convergence without the late fusion overhead.

6.3.2. Cross-Action Space Flow Transformer

We design a novel, efficient Flow Transformer to handle heterogeneous action spaces efficiently. In particular, we introduce *Action-Space Global-AdaLN-Zero*, a unified normalization layer that conditions each transformer block on both temporal signals (e.g., flow time step) and per-action-type embeddings without incurring large parameter overheads, as illustrated in Figure 6.3. Standard AdaLN-Zero uses distinct scale-and-shift parameters per layer, increasing parameter count by 30% in diffusion transformers [148].

In contrast, *Action-Space Global-AdaLN-Zero* shares a single set of modulation weights across all layers, while generating unique modulation signals for each action category (e.g., delta-EEF vs. joint angle), initialized with zeros for stable training. This reduces parameter count by over 20% compared to naive AdaLN-Zero, yet preserves adaptation to action-space statistics. Given this lower per-layer expressiveness, we additionally inject lightweight LoRA adapters into each layer. This provides fine-grained, layer-specific modulation

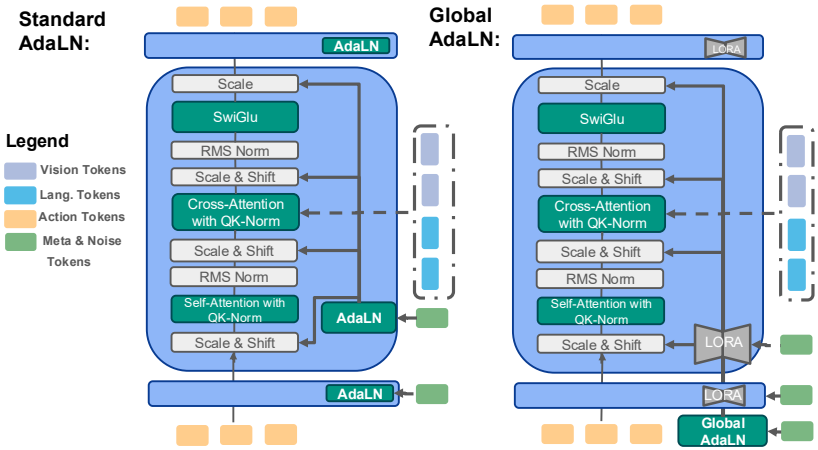


Figure 6.3.: Visualization of AdaLN for Diffusion/Flow Policies. Comparison of standard DiT blocks and our proposed Global AdaLN with layer-specific Lora adapters.

with only a few extra parameters per block. Additionally, each action type uses a small encoder/decoder to map actions into/out of the transformer latent space, enabling consistent handling of differing action dimensions without sacrificing weight sharing.

6.3.3. Rectified Flow for Action Generation

FLOWER uses Rectified Flow for action generation, following the formulation in Section 2.1.4. Flow models learn straight-line velocity fields between noise and data distributions [109, 114], reducing inference computation while maintaining expressiveness—crucial for real-time robot control.

For the conditional action distribution $\pi_\theta(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g}, \boldsymbol{e})$, the flow interpolation is:

$$z_t = (1-t)\boldsymbol{a} + tz_1, \quad z_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{6.1}$$

where $t \in [0, 1]$ is the flow time, $\boldsymbol{a}$ is the ground truth action chunk, and z_1 is standard Gaussian noise. The velocity field v_θ is trained with:

$$\mathcal{L}_{\text{RF}} = \mathbb{E}_{t,z_1}\left[\|(z_1 - \boldsymbol{a}) - v_\theta(z_t, t, \boldsymbol{s}, \boldsymbol{l}, \boldsymbol{e})\|^2\right]. \tag{6.2}$$

Inference requires only $N = 4$ Euler steps for single-arm manipulation and $N = 8$ for high-frequency bimanual tasks.

6.3.4. FLOWER: Efficient Flow-based Vision-Language-Action Models

Leveraging our contributions and insights in intermediate fusion and Global-AdaLN, we present FLOWER: a compact, efficient VLA policy with exceptional performance-to-parameter ratio. FLOWER uses half of the Florence-2-L VLM as its primary backbone, which our ablations show provides optimal performance for robotic manipulation tasks. The model employs an 18-layer Flow Transformer with 1024 latent dimension. Together with the action-specific modules, FLOWER has 947M parameters in total and only requires 1.85 GB of VRAM. This architecture achieves the computational efficiency required for real-time deployment.

Cost-Effective Pretraining Setup. To enable fast, cost-efficient pretraining, FLOWER uses a curated mixture of eight public datasets from OXE (approximately 250k trajectories total). We prioritize broad scene and embodiment diversity—including Franka Pandas and XARMs—and, inspired by Lin et al. [108], draw 75% of our data samples from DROID [85], Google Robot [226], and BridgeV2 [200]. Unlike most OXE datasets, these data are gathered in varied environments with rich distractions, backgrounds, and objects. We use an action chunk length of 20 and a single static image input. The image count is a choice in our pretraining setup, evaluated in Appendix D.2.1. By training on a compact mixture of 74% delta-EEF and 26% single-arm joint-state data, we complete $360,000$ steps in 48 h ($\approx$ 200 GPU-hours). Extending the run yields no further gains (see App. D.2). To identify this optimum, we track zero-shot success on the Bridge SIMPLER benchmark [106], observing that FLOWER Bridge performance stagnates thereafter.

6.4. Evaluation

We structure our evaluation around three research questions:

RQ I How do intermediate fusion, Global AdaLN, and other design choices impact performance?

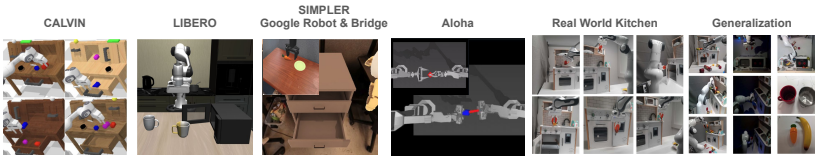


Figure 6.4.: Simulation Environments used to test FLOWER. From left to right: *CALVIN* [130], *LIBERO* [110], *SIMPLER* [106] with the *Bridge* and *Google Robot* variants and *Aloha Simulation Benchmark* [222]. The simulation panels credit the original benchmark papers; the ALOHA Cube Transfer and Bimanual Insertion tasks were introduced with ACT [222]. The final two panels show our own real-world kitchen and generalization settings with clutter, different lighting, and novel objects.

RQ II Does FLOWER match state-of-the-art VLAs while reducing computational cost?

RQ III Can a sub-1B parameter VLA generalize across embodiments, action spaces, and novel environments?

We evaluate across 190 tasks in 10 benchmarks: CALVIN [130], LIBERO [110], SIMPLER [106], Aloha [222], and real-world kitchen manipulation. We use a history length of 1 and an action chunk length of 10 for all single-arm simulation setups, 20 for real-world setup and 50 for the bi-manual finetuning experiments.

6.4.1. Evaluation of Critical Design Decisions for Efficient Flow VLAs

In this section, we evaluate the key design decisions that impact the efficiency and performance of small Flow-based VLAs across two benchmarks: CALVIN ABC [130] and LIBERO-Long [110]. We train each policy variant with 3 seeds for up to $100k$ steps and report the average performance. While CALVIN-ABC focuses on generalization for free form instruction following to solve 34 different task, LIBERO-Long emphasizes long-horizon task completion. Both benchmarks are established for testing VLAs and provide useful insights into optimal design.

Does intermediate fusion provide strong performance with higher efficiency? The fusion strategy determines how the VLM's vision-language

Fusion Strategy	Performance (%)			
	Florence-VLM		SmolVLM2	
	C-ABC	L-Long	C-ABC	L-Long
Early	57.1 ± 5.3	33.4 ± 6.0	25.8 ± 3.9	44.5 ± 2.7
Inter.	**89.5 ± 1.0**	**93.4 ± 2.0**	**72.1 ± 5.0**	**70.7 ± 2.3**
Late	71.2 ± 2.2	61.8 ± 2.5	66.3 ± 2.0	69.2 ± 1.9

Table 6.1.: Evaluation of different VLA fusion strategies. Intermediate fusion yields the best performance across both VLM types. C-ABC reports mean CALVIN chain length divided by 5 and multiplied by 100%; L-Long reports full-task success.

Layers	Model	
	C-ABC	L-Long
Full	66.3 ± 2.0	69.2 ± 1.9
0.2	68.6 ± 3.2	71.8 ± 3.7
0.3	72.1 ± 5.0	70.7 ± 2.3
0.5	66.4 ± 6.4	62.5 ± 3.5

Table 6.2.: Comparing Layer Pruning for CALVIN ABC and LIBERO Long.

features integrate with the action prediction model. We compare three strategies: early fusion (combining noise token with VLM tokens at LLM level), our proposed intermediate fusion, and late fusion (using final VLM outputs to condition the flow prediction module). As shown in Table 6.1 and Table 6.2, intermediate fusion outperforms alternatives with both model variants and on both benchmarks. With Florence-VLM, it achieves 93.4% success on LIBERO-Long, a 60.0 percentage point improvement over early fusion (33.4%) and 31.6 points over late fusion (61.8%). This confirms our hypothesis that intermediate-layer features provide an optimal balance of semantic richness and computational efficiency, addressing **(RQ I)**.

What type of VLM backbone is best for efficient VLAs? We explore different VLM backbones, comparing Florence-2-L [211] and SmolVLM2-500M [126]. Beyond their architectural differences (Florence-2 being encoder-decoder and SmolVLM2 decoder-only), their pretraining objectives significantly differ. Florence-2 was trained on FLD-5B (5.4 billion annotations across 126 million images) with emphasis on object detection, segmentation and visual grounding, while SmolVLM2's pretraining prioritizes general reasoning and language understanding capabilities across text, videos and images. This

Variant	Avg. Len.
FLOWER	**4.44±0.04**
+ standard AdaLN	4.43±0.03
- Flow Head	3.33±0.04
- Custom LR	4.40±0.05
- No VLM train	2.65±0.36
+ small Florence	4.26±0.04
- VLM	3.42±0.07
+ Discrete Token	1.12±0.12
+ Smaller Head	2.60 ±0.09

Table 6.3.: Ablation results for FLOWER on CALVIN ABC.

is representative for a standard VLM pretraining. Florence-2's pretraining transfers more effectively to robotic manipulation tasks, addressing **(RQ I)**. We therefore use the Florence-2 variant of FLOWER for all subsequent experiments.

Does Global-AdaLN enable more efficient Flow Transformers? Next, we compare our proposed Global-AdaLN against default AdaLN used in prior work [163, 84] for our Florence-based FLOWER model on CALVIN ABC. The results in Table 6.3 demonstrate that our proposed Global AdaLN enables relevant parameter reduction of 20% without reducing the performance.

Do we need a large-capacity Diffusion Transformer? Next, we test FLOWER with a small action head using 384 latent dimensions and 6 layers. The final model only achieves significantly lower average performance of 2.60 compared to our design. This confirms that having a high capacity Diffusion Transformer is crucial for performance.

What other design choices matter? Finally, we test our VLA with several ablation variants using the Florence-2 version on CALVIN ABC (Table 6.3). The smaller Florence reduces performance notably, while the frozen VLM has an even bigger negative impact on performance. In addition, our custom Learning Rate scheduler increases performance. Finally, we compare FLOWER with an L1-prediction head like ACT [222] and a discrete token variant to demonstrate the importance of using flow prediction. These findings complete our analysis for **(RQ I)**.

6.4.2. Simulation Experiments

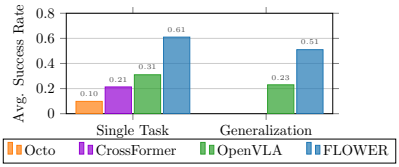
Next, we take the best performing VLA variant, FLOWER, with the Florence-2-L backbone and pretrain it on our pretraining mix. After pretraining, we evaluate FLOWER across multiple simulation benchmarks to assess its performance, generalization capabilities, and adaptation to different robotic environments.

Benchmark Evaluation. We evaluate FLOWER on four established benchmark suites (detailed in Section 2.3.5): **CALVIN** [130] (34 tabletop tasks, three settings: D, ABC, ABCD), **LIBERO** [110] (four 10-task variants plus LIBERO-90), **Aloha** [222] (bimanual manipulation at 50Hz), and **SIMPLER** [106] (real-to-sim transfer evaluation).

Methods	Size	SimplerEnv		Aloha	LIBERO					CALVIN		
		GoogleR	WidowX		Spatial	Object	Goal	Long	90	D	ABC	ABCD
Diff-P-CNN [27]	0.2B	-	-	-	78.3	92.5	68.3	50.5	-	-	1.35	3.16
Seq-BESO [162]	0.2B	-	-	-	-	-	-	-	-	3.19	2.06	4.05
MDT† [163]	0.2B	-	-	-	95.2	97.8	-	83.0	-	3.72	-	4.52
Octo [140]	0.1B	11.0	16.0	-	78.9	85.7	84.6	51.1	-	-	-	-
CrossFormer	0.1B	1.3	30.0	-	-	-	-	-	-	-	-	-
GR-1 [208]	0.2B	-	-	-	-	-	-	-	-	-	3.06	4.21
Seer [193]	0.3B	-	-	-	-	-	-	87.7	-	-	4.28	-
3DDA [84]	0.1B	-	-	-	-	-	-	-	-	-	3.35	-
MoDE [160]	0.4B	-	-	-	-	-	-	94.0	95.0	-	4.01	4.39
RT-1-X [30]	0.5B	42.4	1.1	-	-	-	-	-	-	-	-	-
Baku [59]	-	-	-	-	-	-	-	86.0	90.0	-	-	-
SmolVLA [181]	0.3B	-	-	-	93.0	94.0	91.0	77.0	-	-	-	-
GR00T-N1 [10]	2B	46.7	-	-	94.4	97.6	93.0	90.6	-	-	-	-
π_0 [11]	3B	-	27.8	34	96.8	98.8	95.8	85.2	-	3.76	3.84	-
π_0-FAST [151]	3B	61	39.5	-	96.4	96.8	88.6	60.2	-	-	4.24	4.45
$\pi_{0.5}$-ki [38]	3B	-	-	-	**98.0**	97.8	95.6	85.8	**96.0**	-	-	-
RoboFlamingo [105]	3B	-	-	-	-	-	-	-	-	-	2.47	4.09
SpatialVLA [154]	4B	**72.9**	42.7	-	88.2	89.9	78.6	55.5	-	-	-	-
DeeR-VLA [216]	7B	-	-	-	-	-	-	-	-	-	2.90	4.13
OpenVLA [89]	7B	24.5	1.0	-	84.7	88.4	79.2	53.7	-	-	3.27	-
OpenVLA-OFT [88]	7B	58.5	31.3	-	97.6	98.4	**97.9**	94.5	-	-	-	-
CoA-VLA [102]	7B	-	-	-	85.3	93.1	85.8	55.0	-	-	-	-
FLOWER (Ours)	1B	31.9	**45.0**	**41**	97.5	**99.1**	96.1	**94.9**	94.7	**4.35**	**4.53**	**4.67**

Table 6.4.: Comparison of FLOWER with state-of-the-art VLA methods across SimplerEnv (Google Robot and WidowX), Aloha-Sim, LIBERO, and CALVIN benchmarks. Size indicates total model parameters. CALVIN reports average chain length across D (single environment), ABC (zero-shot generalization), and ABCD (multi-environment) settings. SIMPLER reports full-task success; the RT-1-X, Octo, CrossFormer, OpenVLA, and FLOWER entries follow the detailed FLOWER task tables [161], reproduced in Appendix D.3. Google Robot includes four categories (including apple placement), and retains the reported FLOWER task-table scores; Chapter 5 instead computes an explicit three-category mean across Visual Matching and Variant Aggregation. WidowX excludes partial grasp scores. †MDT LIBERO results are from the OpenVLA-OFT comparison [88]: training from scratch on the original, unfiltered dataset with 100% language annotations (reported for the fully labeled setting), distinct from the 2%-labeled setting in Chapter 4.

Baselines. We compare FLOWER against state-of-the-art VLA policies and specialized approaches, using results reported in prior publications for fair comparison. Our primary comparisons include OpenVLA [89] (7.7B parameters), OpenVLA-OFT [88], the π_0, π_0-FAST, and $\pi_{0.5}$-ki variants [11, 38, 151] (3B parameters), and Diffusion Policy with a CNN backbone [27]. We provide detailed comparisons against additional baselines for each benchmark in Appendix D.3, including video-based VLAs like VPP [69] and specialized policies like Baku [59]. FLOWER uses 950M total parameters, substantially fewer than the 7B-parameter OpenVLA variants; inference speed and memory are



(a) Real World Results for different Generalist Policies finetuned on a Franka Panda Kitchen Setup.

Generalization	FLOWER	OpenVLA
Novel Object	**33.3%**	10.0%
Flashlight	**50.0%**	25.0%
BG Distractors	**69.5%**	41.7%
Tasks Composition	**51.1%**	16.7%
Average	**51.0%**	23.4%

(b) Generalization Results comparing FLOWER and OpenVLA across different scenarios.

Figure 6.5.: Real World Results for Multi-Task and Generalization Experiments

compared in Table 6.5. For reference, finetuning OpenVLA-OFT on a single LIBERO setup requires comparable compute to FLOWER pretraining.

Results. Table 6.4 summarizes the results. FLOWER achieves the highest listed average chain lengths on CALVIN and competitive results on LIBERO and Aloha. It does not lead every benchmark: OpenVLA-OFT has a higher four-suite LIBERO average, and $\pi_{0.5}$-ki has a higher LIBERO-90 score. Importantly, FLOWER was evaluated zero-shot on SIMPLER without task-specific finetuning, whereas OpenVLA-OFT and all π models require full finetuning to achieve their reported results. FLOWER surpasses OpenVLA on CALVIN and LIBERO by wide margins despite its smaller size and faster training, and outperforms π_0 on CALVIN, LIBERO, and Aloha. These results demonstrate FLOWER's versatility in adapting to diverse embodiments and task settings, confirming strong performance at low computational cost **(RQ II)** and generalization to unseen environments **(RQ III)**.

Comparison with Thesis Contributions. On CALVIN, FLOWER improves on BESO (2.06 to 4.53 on ABC and 4.05 to 4.67 on ABCD), MDT (4.52 to 4.67 on ABCD), and MoDE (4.01 to 4.53 on ABC and 4.39 to 4.67 on ABCD). On LIBERO-Long, FLOWER achieves 94.9% compared with MoDE's 94.0%, while MoDE retains the higher LIBERO-90 score (95.0% versus 94.7%). MDT's 83.0% LIBERO-Long result uses the fully language-labeled setting reported by Kim, Finn, and Liang [88], whereas Chapter 4 studies sparse language annotations. FLOWER uses one pretraining recipe followed by separate finetuning for each setting, demonstrating how the architectural advances developed throughout this thesis support a generalist policy **(RQ I, II, III)**.

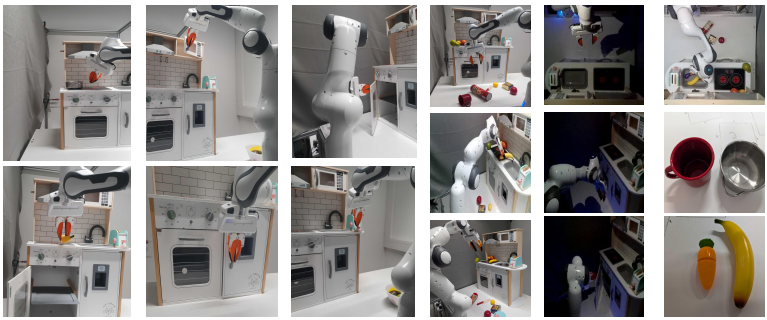
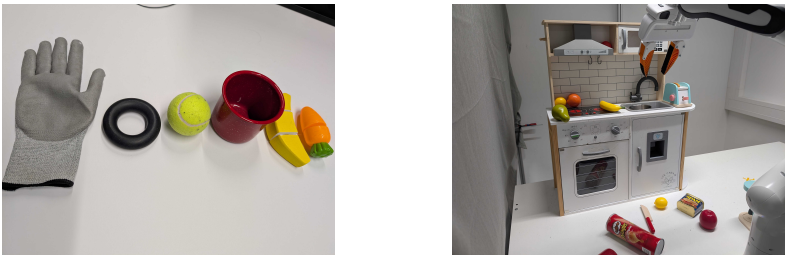


Figure 6.6.: Overview of Real World Tasks and Generalization Settings. The left half shows example tasks used to evaluate policies in the play kitchen setup. The right part shows different challenges of generalization. We test policies in cluttered environments, under reduced lighting and with novel objects.

6.4.3. Real-World Evaluation and Generalization



(a) Unseen Tested Objects

(b) Cluttered Scene with Distractors

Figure 6.7.: Generalization Experiments: Examples of unseen objects (left) and cluttered scenes (right) used to test the adaptability of the policies in our real-world setting. All the tested objects are not included in the training dataset.

Next, we validate our results from simulation in a real world setup. Therefore, we evaluate FLOWER in a real-world kitchen setting with a Franka Panda Robot across 20 distinct tasks involving various objects. The FLOWER dataset contains 417 language-annotated short-horizon segments from approximately 45 minutes of demonstrations. We follow the detailed collection protocol in Appendix D.3.5: a human moves a leader robot to teleoperate the follower, with two static camera views and joint-position actions recorded at 6 Hz.

Method	Throughput (Hz)↑	Latency (s)↓	VRAM (MB)↓
DP (0.26B)	130.67	0.341	**517**
OpenVLA (7B)	6.09	0.164	14574
π_0 (3.3B)	*288.11*	*0.104*	6692
FLOWER+LF(1.15B)	287.36	0.055	2235
FLOWER (0.95B)	**311.04**	**0.052**	*1848*

Table 6.5.: Mean inference efficiency (1000 steps in Bf16). All policies except OpenVLA use chunk length 50. *LF* refers to Late Fusion Ablation of FLOWER with the complete VLM.

This is the same play-data collection used for the BESO experiments in Chapter 3. BESO evaluates a ten-task subset of the kitchen repertoire; FLOWER evaluates all twenty tasks here. MDT uses the separate 4.5-hour partially labeled play collection described in Chapter 4. All Policies are finetuned in joint state space at 6 Hz. An overview of these real-world tasks is shown in Figure 6.6 and we additionally provide rollouts in our project page. We compare FLOWER against several generalist VLA policies: Octo [140], OpenVLA [89], and CrossFormer [36], all finetuned on our dataset using recommended hyperparameters (see Section D.3.5.1 for details).

Results. First, we test all policies on instruction following for all 20 different tasks. Each task is evaluated five times from randomized starting positions. As shown in Figure 6.5a, FLOWER achieves the highest average success rate (61%), approximately twice the success rate of the second-best baseline, OpenVLA (31%). FLOWER demonstrates strong performance compared to state-of-the-art baselines in real-world settings and addresses **(RQ II)**.

Generalization Analysis. We further test FLOWER's generalization capabilities against OpenVLA under challenging conditions: novel objects, flashlight-only lighting, background distractors, and new task compositions. As summarized in Figure 6.5b, FLOWER consistently outperforms OpenVLA across all scenarios, averaging 51.0% success compared to OpenVLA's 23.4%. These experiments verify that FLOWER can effectively adapt to unstructured, real-world variations **(RQ III)**, though it still faces challenges with fine manipulation in highly cluttered environments.

Inference Efficiency. We evaluated FLOWER against baselines on an RTX 4090 GPU (Table 6.5). FLOWER achieves a throughput of 311Hz, making it 8% faster than π_0 (288Hz), and 5007% faster than OpenVLA (6.1Hz). FLOWER has

the lowest memory footprint among VLAs, using 27.6% of π_0's memory and 12.7% of OpenVLA's, making it ideal for commodity hardware. These results strongly support **(RQ II)**.

6.5. Limitations

Despite its advantages, FLOWER still has several limitations. First, it relies on an iterative sampling procedure, which is inherently slower than a single forward pass from deterministic policies. Second, we have validated FLOWER primarily on three manipulation action spaces; its ability to generalize to other embodiments, such as mobile navigation or humanoid locomotion, remains unexplored and is an important direction for future work. Third, pretraining performance for zero-shot deployment on the SIMPLER Google Robot benchmark indicates that further improvements are needed. We hypothesize that the generalization tested in SIMPLER benefits from larger models. Fourth, although FLOWER is considerably smaller than most state-of-the-art VLA models, its $\approx$ 1 B-parameter size may still present deployment challenges in low-resource or real-time settings. Fifth, eight out of our ten used benchmarks are conducted in simulation, limiting the extent to which our results can be taken as evidence of real-world generalization.

6.6. Conclusion

We introduced several contributions for efficient VLAs: intermediate-level fusion that preserves semantic understanding while pruning 30-50% of VLM layers, and global action-space AdaLN that reduces transformer head parameters by 20% without compromising expressivity. These techniques enable compact yet powerful flow-based VLAs. This yields FLOWER, an efficient VLA that matches current state-of-the-art VLAs across 190 tasks in 10 benchmarks, despite requiring only 950M parameters and 200 GPU hours for pretraining. Intermediate modality fusion for VLAs and parameter-efficient conditioning mechanisms democratize access to capable robotics foundation models.

7. Conclusion

In this thesis, we proposed methods to learn multimodal, versatile behavior from partially-labeled, robot-agnostic datasets using behavior cloning. In this chapter, we summarize our main contributions and discuss directions for future work.

7.1. Summary

In Chapter 1, we defined three core challenges for learning versatile behavior from robot- and task-agnostic play data:

C1: Policy Expressivity for Multimodal Behavior. The first challenge focused on policy representation suited for learning from diverse, multimodal play data.

C2: Learning from Heterogeneous, Partially-Labeled Play Data. The second major challenge was data-centric. Many play datasets are only partially labeled and require methods that can extract goal-following capabilities from this setup. Furthermore, robot-agnostic play datasets introduce another core issue: cross-action space and cross-embodiment data.

C3: Computational Efficiency for Training and Deployment. Versatile policies require scaling data and model capacity. However, in contrast to other domains like image generation, runtime and memory requirements are critical for deployment on real robots. Thus, the final challenge concerns efficient policy architectures and training for learning from diverse robot play.

Towards addressing these core challenges, we introduced four contributions:

Chapter 3 proposed score-based diffusion policies as a general policy representation for learning from diverse play data, addressing **C1**. The chapter explored core challenges of adopting score-based diffusion models for policy learning and proposed two novel score-transformer architectures as model backbones. Further, we systematically investigated key design decisions of score-based models for learning actions. We validated BESO's ability to capture multimodal behavior by simultaneously learning goal-conditioned and goal-independent policies via CFG. The classifier-free guidance formulation further enables composable goal-conditioned and goal-independent behaviors from a single model. By predicting clean samples rather than noise and optimizing the noise schedule, BESO achieved state-of-the-art performance in just three denoising steps.

Chapter 4 presented MDT, which extracts language-conditioned policies from partially-labeled play data to tackle **C2**. The core contributions of MDT are two auxiliary objectives for multimodal policy learning from partially-labeled data. CLA introduces a contrastive objective to align the goal-conditioned state representations of diffusion policies, improving multimodal goal following and performance. Further, MGF introduces a foresight objective that enforces the policy to predict masked future frames conditioned on the current goal-conditioned state embedding. Using the insight that synergies exist between generating visual predictions and actions, this objective further boosts policy performance across simulation and real-world settings. Finally, MDT improved BESO's transformer backbone using a lightweight encoder-decoder design. Together, these innovations lead to improved performance in multimodal goal-conditioned learning on diverse benchmarks in simulation and the real world.

Chapter 5 contributed MoDE, the first mixture-of-experts diffusion policy with noise-conditioned expert routing. By treating denoising as a multitask problem, MoDE learns specialist experts for different noise ranges. This design has several advantages: denoising experts can be cached and merged at test time for faster inference with fewer FLOPs. We pretrained MoDE on a cross-embodiment mix of OXE datasets and finetuned it locally on different benchmarks to analyze its scaling ability. Our SIMPLER [106] results are comparable to OpenVLA [89] while using approximately one twentieth of its reported pretraining compute; the Google Robot evaluation follows Fractal finetuning, while WidowX evaluates transfer from pretraining. Our results on LIBERO and CALVIN further verify these findings and demonstrate the advantages of pretraining and expressive policy design.

Chapter 6 unified these insights in FLOWER, a compact and efficient vision-language-action model. FLOWER tackles all challenges addressed in prior chapters: intermediate-fusion conditioning enables faster and more resource-efficient training (**C3**), action-specific Global AdaLN addresses pretraining on heterogeneous play data (**C2**), and rectified flow provides an expressive, multimodal policy representation (**C1**). Pretrained on a cross-action-space dataset in just 200 GPU hours, FLOWER achieved state-of-the-art results across 10 benchmarks with 4 robots and 3 action spaces. On CALVIN ABC→D, it achieves a published average completed-chain length of 4.53 out of 5 instructions (Table D.5). Its generalization is further evaluated in real robot kitchen experiments. Our efficient design democratizes research in large-scale VLA pretraining and enables deployment on edge hardware thanks to its low memory footprint.

7.2. Discussion and Outlook

This thesis offers a general recipe for learning versatile manipulation from partially-labeled robot play: expressive diffusion policies (BESO), self-supervised multimodal alignment (MDT), efficient scaling via noise-conditioned mixture-of-experts (MoDE), and compact vision-language-action models (FLOWER). Together, these contributions demonstrate state-of-the-art performance with compact models and limited compute.

Yet fundamental challenges remain: some with clear research directions, others representing more general open problems in the field of imitation learning.

7.2.1. Follow-Up Research Directions

Dynamics-Aware Visual Representations. MDT's self-supervised objectives partially compensate for CLIP's limitations, but the core issue persists in all modern VLMs: CLIP encodes static image-text correspondences without temporal or dynamics grounding [121]. Robotics requires representations that understand concepts like "pushing" as seen in motion, not frozen frames. Policy finetuning partially bridges this gap but degrades the generalizable visual features learned during pretraining [89]. The release of large-scale

egocentric robot and human video datasets with action labels [68, 85] creates an opportunity to incorporate these datasets with novel loss objectives into the pretraining phase of CLIP. The next generation of VLAs requires a robot-first training paradigm across all stages of foundation models: from CLIP pretraining to VLA finetuning.

Embodied Memory. Current policies, including all image-based architectures in this thesis, condition on single frames only. This suffices for reactive tasks but fails for sequential behaviors requiring state tracking, like scooping three times or remembering which drawer was already checked. Naively concatenating frame histories degrades performance due to domain shift between human demonstrations and robot execution [27]. The field needs memory mechanisms designed for embodied control: compact, robust to distribution shift, and queryable by the policy. Whether this resembles attention over keyframes, explicit state registers, or learned text summarization remains open for future exploration.

7.2.2. Fundamental Open Research Problems in Imitation Learning

What Makes a Good Demonstration? Imitation learning is bounded by demonstration quality, yet we lack any principled understanding of what "quality" means. Is a slow, deliberate demonstration better than a fast one? How much failure recovery data do we need? These questions lack evidence-based scientific answers. Current best practices rely on intuition and many iterations of trial-and-error with few work directly addressing these challenges [108, 7]. Without clear scientific understanding, scaling data collection remains guided by intuition rather than principle.

Data Composition for Transfer. Large-scale VLA pretraining mixes heterogeneous robot datasets, but which combinations yield positive transfer during local finetuning? Current practices are empirical due to the high cost of VLA pretraining. Most papers report single runs without ablations on dataset composition [89, 113]. Future work must develop methods to better understand the interdependencies of robot datasets for principled VLA training paradigms. Since compute costs make systematic ablation infeasible, we need theoretical frameworks or efficient proxies to predict transfer before committing thousands of GPU hours.

7.2.3. Closing Perspective

When I started this thesis four years ago, I believed that algorithmic limitations were the primary bottleneck for achieving general manipulation agents. I now think the core bottleneck is data: its collection, curation, and dataset composition. Diffusion policies combined with VLMs have largely solved the expressivity problem.

FLOWER demonstrates that compact, efficient VLAs can match models requiring orders of magnitude more compute. This matters because accessible research is faster research. If the next breakthrough requires thousand-GPU clusters, only a few labs can participate. If it requires a few nodes and a few hundred GPU hours, the entire community can iterate.

I opened this thesis with a robot commanded to empty a dishwasher. For any kitchen in any home, that task remains unsolved. The methods presented in this thesis make that task achievable in a single environment with sufficient data, providing a foundation for future work to extend them toward the general case.

Bibliography

[1] Anurag Ajay et al. “Is Conditional Generative Modeling all you need for Decision Making?” In: *International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=sP1fo2K9DFG.

[2] Jean-Baptiste Alayrac et al. “Flamingo: a visual language model for few-shot learning”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 23716–23736.

[3] Michael S Albergo and Eric Vanden-Eijnden. “Building normalizing flows with stochastic interpolants”. In: *arXiv preprint arXiv:2209.15571* (2022).

[4] Brenna D Argall et al. “A survey of robot learning from demonstration”. In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[5] Saurabh Arora and Prashant Doshi. “A survey of inverse reinforcement learning: Challenges, methods and progress”. In: *Artificial Intelligence* 297 (2021), p. 103500.

[6] Philipp Becker et al. “Reinforcement Learning from Multiple Sensors via Joint Representations”. In: *arXiv preprint arXiv:2302.05342* (2023).

[7] Suneel Belkhale, Yuchen Cui, and Dorsa Sadigh. “Data quality in imitation learning”. In: *Advances in neural information processing systems* 36 (2023), pp. 80375–80395.

[8] Suneel Belkhale and Dorsa Sadigh. “PLATO: Predicting Latent Affordances Through Object-Centric Play”. In: *6th Annual Conference on Robot Learning*. 2022. URL: https://openreview.net/forum?id=UAA5bNospA0.

[9] Homanga Bharadhwaj et al. *RoboAgent: Generalization and Efficiency in Robot Manipulation via Semantic Augmentations and Action Chunking*. 2023. arXiv: 2309.01918 [cs.RO].

[10] Johan Bjorck et al. “GR00T N1: An Open Foundation Model for Generalist Humanoid Robots”. In: *arXiv preprint arXiv:2503.14734* (2025).

[11] Kevin Black et al. “π_0: A Vision-Language-Action Flow Model for General Robot Control”. In: *Proceedings of Robotics: Science and Systems*. 2025. DOI: 10.15607/RSS.2025.XXI.010. URL: https://www.roboticsproceedings.org/rss21/p010.html.

[12] Kevin Black et al. “Zero-shot robotic manipulation with pretrained image-editing diffusion models”. In: *arXiv preprint arXiv:2310.10639* (2023).

[13] Black Forest Labs et al. “FLUX.1 Kontext: Flow Matching for In-Context Image Generation and Editing in Latent Space”. In: *arXiv preprint arXiv:2506.15742* (2025).

[14] Nils Blank et al. “Scaling Robot Policy Learning via Zero-Shot Labeling with Foundation Models”. In: *8th Annual Conference on Robot Learning*. 2024. URL: https://openreview.net/forum?id=EdVNB2kHv1.

[15] Denis Blessing et al. “Information Maximizing Curriculum: A Curriculum-Based Approach for Learning Versatile Skills”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. Vol. 36. 2023. URL: https://openreview.net/forum?id=7eW6NzSE4g.

[16] Konstantinos Bousmalis et al. “RoboCat: A self-improving foundation agent for robotic manipulation”. In: *arXiv preprint arXiv:2306.11706* (2023).

[17] Max Braun et al. “Riemannian Flow Matching Policy for Robot Motion Learning”. In: *arXiv preprint arXiv:2403.10672* (2024).

[18] Anthony Brohan et al. “RT-1: Robotics Transformer for Real-World Control at Scale”. In: *arXiv preprint arXiv:2212.06817*. 2022.

[19] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. “InstructPix2Pix: Learning to Follow Image Editing Instructions”. In: *CVPR*. 2023.

[20] Qingwen Bu et al. “Towards Synergistic, Generalized, and Efficient Dual-System for Robotic Manipulation”. In: *arXiv preprint arXiv:2410.08001* (2024).

[21] Remi Cadene et al. *LeRobot: State-of-the-art Machine Learning for Real-World Robotics in Pytorch*. https://github.com/huggingface/lerobot. 2024.

[22] Onur Celik, Aleksandar Taranovic, and Gerhard Neumann. "Acquiring Diverse Skills using Curriculum Reinforcement Learning with Mixture of Experts". In: *arXiv preprint arXiv:2403.06966* (2024).

[23] Onur Celik et al. "Specializing Versatile Skill Libraries using Local Mixture of Experts". In: *Proceedings of the 5th Conference on Robot Learning* (Nov. 8–11, 2022). Ed. by Aleksandra Faust, David Hsu, and Gerhard Neumann. Vol. 164. Proceedings of Machine Learning Research. PMLR, Nov. 2022, pp. 1423–1433. URL: https://proceedings.mlr.press/v164/celik22a.html.

[24] Chi-Lam Cheang et al. "GR-2: A Generative Video-Language-Action Model with Web-Scale Knowledge for Robot Manipulation". In: *arXiv preprint arXiv:2410.06158* (2024).

[25] Lili Chen, Shikhar Bahl, and Deepak Pathak. "PlayFusion: Skill Acquisition via Diffusion from Language-Annotated Play". In: *7th Annual Conference on Robot Learning*. 2023.

[26] Tao Chen, Adithyavairavan Murali, and Abhinav Gupta. "Hardware conditioned policies for multi-robot transfer learning". In: *Advances in Neural Information Processing Systems* 31 (2018).

[27] Cheng Chi et al. "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2023.

[28] Cheng Chi et al. "Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots". In: *arXiv preprint arXiv:2402.10329* (2024).

[29] Zewen Chi et al. "On the Representation Collapse of Sparse Mixture of Experts". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=mWaYC6CZf5.

[30] Open X-Embodiment Collaboration. *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*. https://arxiv.org/abs/2310.08864. 2023.

[31] Zichen Jeff Cui et al. "From Play to Policy: Conditional Behavior Generation from Uncurated Robot Data". In: *International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=c7rM7F7jQjN.

[32] Sudeep Dasari et al. “The Ingredients for Robotic Diffusion Transformers”. In: *arXiv preprint arXiv:2410.10088* (2024).

[33] Matt Deitke et al. “Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models”. In: *Proceedings of the Computer Vision and Pattern Recognition Conference*. 2025, pp. 91–104.

[34] Prafulla Dhariwal and Alexander Nichol. “Diffusion models beat gans on image synthesis”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 8780–8794.

[35] Yiming Ding et al. “Goal-conditioned imitation learning”. In: *Advances in neural information processing systems* 32 (2019).

[36] Ria Doshi et al. “Scaling Cross-Embodied Learning: One Policy for Manipulation, Navigation, Locomotion and Aviation”. In: *8th Annual Conference on Robot Learning*. 2024.

[37] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations*. 2020.

[38] Danny Driess et al. “Knowledge insulating vision-language-action models: Train fast, run fast, generalize better”. In: *arXiv preprint arXiv:2505.23705* (2025).

[39] Danny Driess et al. “PaLM-E: An embodied multimodal language model”. In: *arXiv preprint arXiv:2303.03378* (2023).

[40] Nan Du et al. “GLaM: Efficient Scaling of Language Models with Mixture-of-Experts”. In: *Proceedings of the 39th International Conference on Machine Learning* (July 17–23, 2022). Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 5547–5569. arXiv: 2112.06905 [cs.CL]. URL: https://proceedings.mlr.press/v162/du22c.html.

[41] Yilun Du et al. “Learning universal policies via text-guided video generation”. In: *arXiv preprint arXiv:2302.00111* (2023).

[42] Patrick Esser et al. “Scaling rectified flow transformers for high-resolution image synthesis”. In: *Forty-first International Conference on Machine Learning*. 2024.

[43] Haritheja Etukuru et al. “Robot utility models: General policies for zero-shot deployment in new environments”. In: *arXiv preprint arXiv:2409.05865* (2024).

[44] Benjamin Eysenbach et al. "Contrastive Learning as Goal-Conditioned Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. 2022. URL: https://openreview.net/forum?id=vGQiU5sqUe3.

[45] Benjamin Eysenbach et al. "Imitating Past Successes can be Very Suboptimal". In: *Advances in Neural Information Processing Systems*. 2022. URL: https://openreview.net/forum?id=iqC03jbPjYF.

[46] William Fedus, Barret Zoph, and Noam Shazeer. "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity". In: *Journal of Machine Learning Research* 23.120 (2022), pp. 1–39.

[47] Pete Florence et al. "Implicit behavioral cloning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 158–168.

[48] Justin Fu, Katie Luo, and Sergey Levine. "Learning Robust Rewards with Adversarial Inverse Reinforcement Learning". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rkHywl-A-.

[49] Zipeng Fu, Tony Z Zhao, and Chelsea Finn. "Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation". In: *arXiv preprint arXiv:2401.02117* (2024).

[50] Niklas Funk et al. "ActionFlow: Equivariant, Accurate, and Efficient Policies with Spatially Symmetric Flow Matching". In: *arXiv preprint arXiv:2409.04576* (2024).

[51] Xinyang Geng et al. "Multimodal masked autoencoders learn transferable representations". In: *arXiv preprint arXiv:2205.14204* (2022).

[52] Nikolaos Gkanatsios et al. "Energy-based Models are Zero-Shot Planners for Compositional Scene Rearrangement". In: *Robotics: Science and Systems*. 2023.

[53] Hyojun Go et al. "Towards practical plug-and-play diffusion models". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2023, pp. 1962–1971.

[54] Ian J Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[55] Jean-Bastien Grill et al. "Bootstrap your own latent-a new approach to self-supervised learning". In: *Advances in neural information processing systems* 33 (2020), pp. 21271–21284.

[56] Jiayuan Gu et al. “RT-Trajectory: Robotic Task Generalization via Hindsight Trajectory Sketches”. In: *International Conference on Learning Representations*. 2024.

[57] Abhishek Gupta et al. “Relay Policy Learning: Solving Long Horizon Tasks via Imitation and Reinforcement Learning”. In: *Conference on Robot Learning (CoRL)* (2019).

[58] Huy Ha, Pete Florence, and Shuran Song. “Scaling Up and Distilling Down: Language-Guided Robot Skill Acquisition”. In: *7th Annual Conference on Robot Learning*. 2023.

[59] Siddhant Haldar, Zhuoran Peng, and Lerrel Pinto. “BAKU: An Efficient Transformer for Multi-Task Policy Learning”. In: *arXiv preprint arXiv:2406.07539* (2024).

[60] Tiankai Hang et al. *Efficient Diffusion Training via Min-SNR Weighting Strategy*. 2024. arXiv: 2303.09556 [cs.CV].

[61] Kay Hansel et al. “Hierarchical policy blending as inference for reactive robot control”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 10181–10188.

[62] Hussein Hazimeh et al. “DSelect-k: Differentiable Selection in the Mixture of Experts with Applications to Multi-Task Learning”. In: *CoRR* abs/2106.03760 (2021). arXiv: 2106.03760. URL: https://arxiv.org/abs/2106.03760.

[63] Kaiming He et al. “Masked autoencoders are scalable vision learners”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 16000–16009.

[64] Alex Henry et al. “Query-key normalization for transformers”. In: *arXiv preprint arXiv:2010.04245* (2020).

[65] Jonathan Ho and Stefano Ermon. “Generative adversarial imitation learning”. In: *Advances in neural information processing systems* 29 (2016).

[66] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6840–6851.

[67] Jonathan Ho and Tim Salimans. “Classifier-Free Diffusion Guidance”. In: *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*. 2021.

[68] Ryan Hoque et al. "EgoDex: Learning Dexterous Manipulation from Large-Scale Egocentric Video". In: *arXiv preprint arXiv:2505.11709* (2025).

[69] Yucheng Hu et al. *Video Prediction Policy: A Generalist Robot Policy with Predictive Visual Representations*. 2024. arXiv: 2412.14803 [cs.CV]. URL: https://arxiv.org/abs/2412.14803.

[70] Wenlong Huang, Igor Mordatch, and Deepak Pathak. "One policy to control them all: Shared modular policies for agent-agnostic control". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4455–4464.

[71] Aapo Hyvärinen. "Estimation of Non-Normalized Statistical Models by Score Matching". In: *Journal of Machine Learning Research* 6.24 (2005), pp. 695–709. URL: https://www.jmlr.org/papers/v6/hyvarinen05a.html.

[72] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[73] Abhinav Jain and Vaibhav Unhelkar. "GO-DICE: Goal-Conditioned Option-Aware Offline Imitation Learning via Stationary Distribution Correction Estimation". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 38. 11. 2024, pp. 12763–12772.

[74] Ayush Jain et al. "Bottom up top down detection transformers for language grounding in images and point clouds". In: *European Conference on Computer Vision*. Springer. 2022, pp. 417–433.

[75] Stephen James et al. "RLBench: The Robot Learning Benchmark & Learning Environment". In: *IEEE Robotics and Automation Letters* (2020).

[76] Michael Janner et al. "Planning with Diffusion for Flexible Behavior Synthesis". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 9902–9915.

[77] Xiaogang Jia et al. "PointMapPolicy: Structured Point Cloud Processing for Multi-Modal Imitation Learning". In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025. URL: https://openreview.net/forum?id=ZR2mdBrhJX.

[78] Xiaogang Jia et al. “Towards Diverse Behaviors: A Benchmark for Imitation Learning with Human Demonstrations”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=6pPYRXKPpw.

[79] Albert Q Jiang et al. “Mixtral of experts”. In: *arXiv preprint arXiv:2401.04088* (2024).

[80] Xinkai Jiang et al. “A Comprehensive User Study on Augmented Reality-Based Data Collection Interfaces for Robot Learning”. In: *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*. 2024, pp. 333–342.

[81] Yunfan Jiang et al. “VIMA: General Robot Manipulation with Multimodal Prompts”. In: *Fortieth International Conference on Machine Learning*. 2023.

[82] Siddharth Karamcheti et al. “Language-driven representation learning for robotics”. In: *arXiv preprint arXiv:2302.12766* (2023).

[83] Tero Karras et al. “Elucidating the Design Space of Diffusion-Based Generative Models”. In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=k7FuTOWMOc7.

[84] Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. “3D Diffuser Actor: Policy Diffusion with 3D Scene Representations”. In: *8th Annual Conference on Robot Learning*. 2024. URL: https://openreview.net/forum?id=gqCQxObVz2.

[85] Alexander Khazatsky et al. “DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset”. In: *Proceedings of Robotics: Science and Systems*. 2024. DOI: 10.15607/RSS.2024.XX.120. URL: https://www.roboticsproceedings.org/rss20/p120.html.

[86] Junsu Kim et al. “Imitating Graph-Based Planning with Goal-Conditioned Policies”. In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=6lUEy1J5R7p.

[87] Kuno Kim et al. “Imitation with neural density models”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 5360–5372.

[88] Moo Jin Kim, Chelsea Finn, and Percy Liang. “Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success”. In: *arXiv preprint arXiv:2502.19645* (2025).

[89] Moo Jin Kim et al. "OpenVLA: An Open-Source Vision-Language-Action Model". In: *Proceedings of The 8th Conference on Robot Learning*. Vol. 270. Proceedings of Machine Learning Research. PMLR, 2025, pp. 2679–2713. URL: https://proceedings.mlr.press/v270/kim25c.html.

[90] Kimi Team, Yifan Bai, et al. "Kimi K2: Open Agentic Intelligence". In: *arXiv preprint arXiv:2507.20534* (2025). URL: https://arxiv.org/abs/2507.20534.

[91] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[92] Po-Chen Ko et al. "Learning to Act from Actionless Video through Dense Correspondences". In: *arXiv:2310.08576* (2023).

[93] Chieh-Hsin Lai et al. "The Principles of Diffusion Models". In: *arXiv preprint arXiv:2510.21890* (2025).

[94] Hugo Larochelle and Iain Murray. "The neural autoregressive distribution estimator". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 29–37.

[95] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. "Curl: Contrastive unsupervised representations for reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5639–5650.

[96] An Thai Le et al. "Hierarchical policy blending as optimal transport". In: *Learning for Dynamics and Control Conference*. PMLR. 2023, pp. 797–812.

[97] Yann LeCun et al. "A tutorial on energy-based learning". In: *Predicting structured data* 1.0 (2006).

[98] Seungjae Lee et al. "Behavior Generation with Latent Actions". In: *arXiv preprint arXiv:2403.03181* (2024).

[99] Youngwoon Lee et al. "Generalizable imitation learning from observation via inferring goal proximity". In: *Advances in neural information processing systems* 34 (2021), pp. 16118–16130.

[100] Yunsung Lee et al. "Multi-architecture multi-expert diffusion models". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 2024, pp. 13427–13436.

[101] Mike Lewis et al. *BASE Layers: Simplifying Training of Large, Sparse Models*. 2021. arXiv: 2103.16716 [cs.CL].

[102] Jinming Li et al. "Improving Vision-Language-Action Models via Chain-of-Affordance". In: *arXiv preprint arXiv:2412.20451* (2024).

[103] Maximilian Xiling Li et al. "Curriculum-Based Imitation of Versatile Skills". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 2951–2957. DOI: 10.1109/ICRA48891.2023.10160543.

[104] Xiang Li et al. "Crossway Diffusion: Improving Diffusion-based Visuomotor Policy via Self-supervised Learning". In: *arXiv preprint arXiv:2307.01849* (2023).

[105] Xinghang Li et al. "Vision-Language Foundation Models as Effective Robot Imitators". In: *arXiv preprint arXiv:2311.01378* (2023).

[106] Xuanlin Li et al. "Evaluating Real-World Robot Manipulation Policies in Simulation". In: *Proceedings of The 8th Conference on Robot Learning*. Vol. 270. Proceedings of Machine Learning Research. PMLR, 2025, pp. 3705–3728. URL: https://proceedings.mlr.press/v270/li25c.html.

[107] Shalev Lifshitz et al. "STEVE-1: A Generative Model for Text-to-Behavior in Minecraft". In: *arXiv preprint arXiv:2306.00937* (2023).

[108] Fanqi Lin et al. "Data Scaling Laws in Imitation Learning for Robotic Manipulation". In: *The Thirteenth International Conference on Learning Representations*. 2025.

[109] Yaron Lipman et al. "Flow matching for generative modeling". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=PqvMRDCJT9t.

[110] Bo Liu et al. "LIBERO: Benchmarking knowledge transfer for lifelong robot learning". In: *Advances in Neural Information Processing Systems* 36 (2023).

[111] Fanfan Liu et al. "RoboUniView: Visual-Language Model with Unified View Representation for Robotic Manipulation". In: *arXiv preprint 2406.18977* (2024).

[112] Haotian Liu et al. "Visual instruction tuning". In: *Advances in neural information processing systems* 36 (2023), pp. 34892–34916.

[113] Songming Liu et al. “RDT-1B: a Diffusion Foundation Model for Bimanual Manipulation”. In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=yAzN4tz7oI.

[114] Xingchao Liu, Chengyue Gong, and Qiang Liu. “Flow straight and fast: Learning to generate and transfer data with rectified flow”. In: *arXiv preprint arXiv:2209.03003* (2022).

[115] Cheng Lu et al. “DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps”. In: *Advances in Neural Information Processing Systems*. 2022. URL: https://openreview.net/forum?id=2uAaGwlP_V.

[116] Cheng Lu et al. “DPM-Solver++: Fast Solver for Guided Sampling of Diffusion Probabilistic Models”. In: *arXiv preprint arXiv:2211.01095* (2022).

[117] Corey Lynch and Pierre Sermanet. “Language conditioned imitation learning over unstructured data”. In: *arXiv preprint arXiv:2005.07648* (2020).

[118] Corey Lynch et al. “Interactive language: Talking to robots in real time”. In: *IEEE Robotics and Automation Letters* (2023).

[119] Corey Lynch et al. “Learning latent plans from play”. In: *Conference on robot learning*. PMLR. 2020, pp. 1113–1132. arXiv: 1903.01973 [cs.RO].

[120] Nanye Ma et al. “SiT: Exploring Flow and Diffusion-based Generative Models with Scalable Interpolant Transformers”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2024.

[121] Yecheng Jason Ma et al. “LIV: Language-Image Representations and Rewards for Robotic Control”. In: *arXiv preprint arXiv:2306.00958* (2023).

[122] Yecheng Jason Ma et al. “Offline Goal-Conditioned Reinforcement Learning via f-Advantage Regression”. In: *Thirty-Sixth Conference on Neural Information Processing Systems*. 2022. URL: https://openreview.net/forum?id=_h29VprPHD.

[123] Yecheng Jason Ma et al. “VIP: Towards Universal Visual Reward and Representation via Value-Implicit Pre-Training”. In: *The Eleventh International Conference on Learning Representations*. 2022.

[124] Arjun Majumdar et al. “Where are we in the search for an Artificial Visual Cortex for Embodied Intelligence?” In: *Workshop on Reincarnating Reinforcement Learning at ICLR 2023*. 2023. URL: https://openreview.net/forum?id=NJtSbIWmt2T.

[125] Ajay Mandlekar et al. “GTI: Learning to Generalize across Long-Horizon Tasks from Human Demonstrations”. In: *Proceedings of Robotics: Science and Systems*. July 2020. DOI: 10.15607/RSS.2020.XVI.061.

[126] Andrés Marafioti et al. “SmolVLM: Redefining small and efficient multimodal models”. In: *arXiv preprint arXiv:2504.05299* (2025).

[127] Oier Mees, Jessica Borja-Diaz, and Wolfram Burgard. “Grounding language with visual affordances over unstructured data”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 11576–11582.

[128] Oier Mees, Andreas Eitel, and Wolfram Burgard. “Choosing smartly: Adaptive multimodal fusion for object detection in changing environments”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 151–156.

[129] Oier Mees, Lukas Hermann, and Wolfram Burgard. “What Matters in Language Conditioned Robotic Imitation Learning Over Unstructured Data”. In: *IEEE Robotics and Automation Letters (RA-L)* 7.4 (2022), pp. 11205–11212.

[130] Oier Mees et al. “CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks”. In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 7327–7334.

[131] Atharva Mete et al. “QueST: Self-Supervised Skill Abstractions for Learning Continuous Control”. In: *arXiv preprint arXiv:2407.15840* (2024).

[132] MiniMax, Aonian Li, Bangwei Gong, et al. “MiniMax-01: Scaling Foundation Models with Lightning Attention”. In: *arXiv preprint arXiv:2501.08313* (2025). URL: https://arxiv.org/abs/2501.08313.

[133] Suvir Mirchandani et al. “Large Language Models as General Pattern Machines”. In: *Conference on Robot Learning*. PMLR. 2023, pp. 2498–2518.

[134] Niklas Muennighoff et al. “OLMoE: Open Mixture-of-Experts Language Models”. In: *arXiv preprint arXiv:2409.02060* (2024).

[135] Vivek Myers et al. “Goal Representations for Instruction Following: A Semi-Supervised Language Interface to Control”. In: *7th Annual Conference on Robot Learning*. 2023.

[136] Suraj Nair et al. “R3M: A Universal Visual Representation for Robot Manipulation”. In: *6th Annual Conference on Robot Learning*. 2022. URL: https://openreview.net/forum?id=tGbpgz6yOrI.

[137] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[138] Alexander Quinn Nichol et al. “GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models”. In: *Proceedings of the 39th International Conference on Machine Learning* (July 17–23, 2022). Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 16784–16804.

[139] Johan Obando-Ceron et al. “Mixtures of Experts Unlock Parameter Scaling for Deep RL”. In: *arXiv preprint arXiv:2402.08609* (2024).

[140] Octo Model Team et al. “Octo: An Open-Source Generalist Robot Policy”. In: *Proceedings of Robotics: Science and Systems*. 2024. DOI: 10.15607/RSS.2024.XX.090. URL: https://www.roboticsproceedings.org/rss20/p090.html.

[141] OpenAI. *GPT-5 System Card*. Tech. rep. Version updated August 13, 2025; PDF available at https://cdn.openai.com/gpt-5-system-card.pdf. OpenAI, Aug. 2025.

[142] Takayuki Osa et al. “An algorithmic perspective on imitation learning”. In: *Foundations and Trends® in Robotics* 7.1-2 (2018), pp. 1–179.

[143] Jyothish Pari et al. “The Surprising Effectiveness of Representation Learning for Visual Imitation”. In: *Proceedings of Robotics: Science and Systems*. New York City, NY, USA, June 2022. DOI: 10.15607/RSS.2022.XVIII.010. arXiv: 2112.01511 [cs.RO].

[144] Byeongjun Park et al. “Denoising task routing for diffusion models”. In: *arXiv preprint arXiv:2310.07138* (2023).

[145] Byeongjun Park et al. “Switch Diffusion Transformer: Synergizing Denoising Tasks with Sparse Mixture-of-Experts”. In: *arXiv preprint arXiv:2403.09176* (2024).

[146] Austin Patel and Shuran Song. "GET-Zero: Graph Embodiment Transformer for Zero-shot Embodiment Generalization". In: *arXiv preprint arXiv:2407.15002* (2024).

[147] Tim Pearce et al. "Imitating Human Behaviour with Diffusion Models". In: *International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=Pv1GPQzRrC8.

[148] William Peebles and Saining Xie. "Scalable diffusion models with transformers". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 4195–4205.

[149] Ethan Perez et al. "FiLM: Visual Reasoning with a General Conditioning Layer". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[150] Karl Pertsch, Youngwoon Lee, and Joseph Lim. "Accelerating reinforcement learning with learned skill priors". In: *Conference on robot learning*. PMLR. 2021, pp. 188–204.

[151] Karl Pertsch et al. "FAST: Efficient action tokenization for vision-language-action models". In: *arXiv preprint arXiv:2501.09747* (2025).

[152] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 3406–3413.

[153] Dean Pomerleau. "ALVINN: An Autonomous Land Vehicle In a Neural Network". In: *Proceedings of (NeurIPS) Neural Information Processing Systems*. Ed. by D.S. Touretzky. Vol. 1. Morgan Kaufmann, Dec. 1989, pp. 305–313.

[154] Delin Qu et al. "SpatialVLA: Exploring Spatial Representations for Visual-Language-Action Model". In: *arXiv preprint arXiv:2501.15830* (2025).

[155] Qwen Team. "Qwen2.5 Technical Report". In: *arXiv preprint arXiv:2412.15115* (2024).

[156] Alec Radford et al. "Learning Transferable Visual Models From Natural Language Supervision". In: *Proceedings of the 38th International Conference on Machine Learning* (July 18–24, 2021). Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR. PMLR, July 2021, pp. 8748–8763.

[157] Ilija Radosavovic et al. “Real-world robot learning with masked visual pre-training”. In: *Conference on Robot Learning*. PMLR. 2023, pp. 416–426. URL: https://openreview.net/forum?id=KWCZfuqshd.

[158] Krishan Rana, Andrew Melnik, and Niko Sünderhauf. “Contrastive Language, Action, and State Pre-training for Robot Learning”. In: *arXiv preprint arXiv:2304.10782* (2023).

[159] Krishan Rana et al. “SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning”. In: *7th Annual Conference on Robot Learning*. 2023. URL: https://openreview.net/forum?id=wMpOMO0Ss7a.

[160] Moritz Reuss et al. “Efficient Diffusion Transformer Policies with Mixture of Expert Denoisers for Multitask Learning”. In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=nDmwloEl3N.

[161] Moritz Reuss et al. “FLOWER: Democratizing Generalist Robot Policies with Efficient Vision-Language-Flow Models”. In: *Proceedings of The 9th Conference on Robot Learning*. Vol. 305. Proceedings of Machine Learning Research. PMLR, 2025, pp. 3736–3761. URL: https://proceedings.mlr.press/v305/reuss25a.html.

[162] Moritz Reuss et al. “Goal Conditioned Imitation Learning using Score-based Diffusion Policies”. In: *Robotics: Science and Systems*. 2023.

[163] Moritz Reuss et al. “Multimodal Diffusion Transformer: Learning Versatile Behavior from Multimodal Goals”. In: *Robotics: Science and Systems*. 2024.

[164] Carlos Riquelme et al. *Scaling Vision with Sparse Mixture of Experts*. 2021. arXiv: 2106.05974 [cs.CV].

[165] Stephen Roller et al. “Hash Layers For Large Sparse Models”. In: *CoRR* abs/2106.04426 (2021). arXiv: 2106.04426. URL: https://arxiv.org/abs/2106.04426.

[166] Robin Rombach et al. “High-resolution image synthesis with latent diffusion models”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695.

[167] Erick Rosete-Beas et al. “Latent Plans for Task-Agnostic Offline Reinforcement Learning”. In: *6th Annual Conference on Robot Learning*. 2022. URL: https://openreview.net/forum?id=ViYLaruFwN3.

[168] Tim Salimans and Jonathan Ho. "Progressive Distillation for Fast Sampling of Diffusion Models". In: *International Conference on Learning Representations*. 2021.

[169] Stefan Schaal. "Is imitation learning the route to humanoid robots?" In: *Trends in cognitive sciences* 3.6 (1999), pp. 233–242.

[170] Paul Maria Scheikl et al. "Movement Primitive Diffusion: Learning Gentle Robotic Manipulation of Deformable Objects". In: *arXiv preprint arXiv:2312.10008* (2023).

[171] Younggyo Seo et al. "Multi-view masked world models for visual robotic manipulation". In: *arXiv preprint arXiv:2302.02408* (2023).

[172] Nur Muhammad Mahi Shafiullah et al. "Behavior Transformers: Cloning k modes with one stone". In: *Thirty-Sixth Conference on Neural Information Processing Systems*. 2022. URL: https://openreview.net/forum?id=agTr-vRQsa.

[173] Nur Muhammad Mahi Shafiullah et al. "On Bringing Robots Home". In: *arXiv preprint arXiv:2311.16098* (2023).

[174] Rutav Shah, Roberto Martín-Martín, and Yuke Zhu. "MUTEX: Learning Unified Policies from Multimodal Task Specifications". In: *Conference on Robot Learning*. PMLR. 2023, pp. 2663–2682.

[175] Zhihong Shao et al. "DeepSeekMath: Pushing the limits of mathematical reasoning in open language models". In: *arXiv preprint arXiv:2402.03300* (2024).

[176] Noam Shazeer. "GLU variants improve transformer". In: *arXiv preprint arXiv:2002.05202* (2020).

[177] Noam Shazeer et al. *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer*. 2017. arXiv: 1701.06538 [cs.LG].

[178] Yide Shentu et al. "From LLMs to Actions: Latent Codes as Bridges in Hierarchical Robot Control". In: *arXiv preprint arXiv:2405.04798* (2024).

[179] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. "CLIPort: What and where pathways for robotic manipulation". In: *Conference on Robot Learning*. PMLR. 2022, pp. 894–906.

[180] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. "Perceiver-actor: A multi-task transformer for robotic manipulation". In: *Conference on Robot Learning*. PMLR. 2023, pp. 785–799.

[181] Mustafa Shukor et al. “SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics”. In: *arXiv preprint arXiv:2506.01844* (2025).

[182] Avi Singh et al. “Parrot: Data-Driven Behavioral Priors for Reinforcement Learning”. In: *International Conference on Learning Representations*. 2020.

[183] Jascha Sohl-Dickstein et al. “Deep unsupervised learning using nonequilibrium thermodynamics”. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 2256–2265.

[184] Jiaming Song, Chenlin Meng, and Stefano Ermon. “Denoising Diffusion Implicit Models”. In: *ICLR*. 2021.

[185] Yang Song and Stefano Ermon. “Generative modeling by estimating gradients of the data distribution”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[186] Yang Song and Stefano Ermon. “Improved techniques for training score-based generative models”. In: *Advances in neural information processing systems* 33 (2020), pp. 12438–12448.

[187] Yang Song et al. “Consistency Models”. In: *arXiv preprint arXiv:2303.01469* (2023).

[188] Yang Song et al. “Score-Based Generative Modeling through Stochastic Differential Equations”. In: *International Conference on Learning Representations*. 2021.

[189] Austin Stone et al. “Open-World Object Manipulation using Pre-Trained Vision-Language Model”. In: *arXiv preprint*. 2023.

[190] Jianlin Su et al. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. 2021. arXiv: `2104.09864 [cs.CL]`.

[191] Priya Sundaresan et al. “RT-Sketch: Goal-Conditioned Imitation Learning from Hand-Drawn Sketches”. In: *arXiv preprint arXiv:2403.02709* (2024). URL: `https://arxiv.org/abs/2403.02709`.

[192] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[193] Yang Tian et al. “Predictive Inverse Dynamics Models are Scalable Learners for Robotic Manipulation”. In: *arXiv preprint arXiv:2412.15109* (2024). URL: `https://arxiv.org/abs/2412.15109`.

[194] Hugo Touvron et al. “LLaMA: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023).

[195] Julen Urain et al. “Se (3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 5923–5930.

[196] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. “Pixel recurrent neural networks”. In: *International conference on machine learning*. PMLR. 2016, pp. 1747–1756.

[197] Francisco Vargas, Will Sussman Grathwohl, and Arnaud Doucet. “Denoising Diffusion Samplers”. In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=8pvnfTAbu1f.

[198] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[199] Pascal Vincent. “A Connection Between Score Matching and Denoising Autoencoders”. In: *Neural Computation* 23.7 (2011), pp. 1661–1674. DOI: 10.1162/NECO_a_00142.

[200] Homer Rich Walke et al. “Bridgedata v2: A dataset for robot learning at scale”. In: *Conference on Robot Learning*. PMLR. 2023, pp. 1723–1736.

[201] Wan Team. “Wan: Open and Advanced Large-Scale Video Generative Models”. In: *arXiv preprint arXiv:2503.20314* (2025).

[202] Chen Wang et al. “MimicPlay: Long-Horizon Imitation Learning by Watching Human Play”. In: *7th Annual Conference on Robot Learning*. 2023. URL: https://openreview.net/forum?id=hRZ1YjDZmTo.

[203] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. “Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning”. In: *International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=AHvFDPi-FA.

[204] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *Advances in neural information processing systems* 35 (2022), pp. 24824–24837.

[205] Jason Wei et al. “Emergent abilities of large language models”. In: *arXiv preprint arXiv:2206.07682* (2022).

[206] Junjie Wen et al. “TinyVLA: Towards fast, data-efficient vision-language-action models for robotic manipulation”. In: *arXiv preprint arXiv:2409.12514* (2024).

[207] Tim Windecker et al. “NaviTrace: Evaluating Embodied Navigation of Vision-Language Models”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2026.

[208] Hongtao Wu et al. “Unleashing Large-Scale Video Generative Pre-training for Visual Robot Manipulation”. In: *International Conference on Learning Representations*. 2024.

[209] Zhou Xian et al. “ChainedDiffuser: Unifying Trajectory Diffusion and Keypose Prediction for Robotic Manipulation”. In: *7th Annual Conference on Robot Learning*. 2023. URL: https://openreview.net/forum?id=W0zgY2mBTA8.

[210] Zhou Xian et al. “Unifying Diffusion Models with Action Detection Transformers for Multi-task Robotic Manipulation”. In: *7th Annual Conference on Robot Learning*. 2023.

[211] Bin Xiao et al. “Florence-2: Advancing a unified representation for a variety of vision tasks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 4818–4829.

[212] Ted Xiao et al. “Robotic skill acquisition via instruction augmentation with vision-language models”. In: *arXiv preprint arXiv:2211.11736* (2022).

[213] Tete Xiao et al. “Masked visual pre-training for motor control”. In: *arXiv preprint arXiv:2203.06173* (2022).

[214] Jonathan Heewon Yang, Dorsa Sadigh, and Chelsea Finn. “Polybot: Training One Policy Across Robots While Embracing Variability”. In: *Conference on Robot Learning*. PMLR. 2023, pp. 2955–2974.

[215] Sarah Young et al. “Playful Interactions for Representation Learning”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 992–999. DOI: 10.1109/IROS47612.2022.9981307.

[216] Yang Yue et al. “DeeR-VLA: Dynamic Inference of Multimodal Large Language Models for Efficient Robot Execution”. In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 56619–56643.

[217] Xiaohua Zhai et al. “Sigmoid loss for language image pre-training”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 11975–11986.

[218] Albert Zhan et al. “Learning visual robotic control efficiently with contrastive pre-training and data augmentation”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 4040–4047.

[219] Biao Zhang and Rico Sennrich. “Root mean square layer normalization”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[220] Edwin Zhang et al. “Language Control Diffusion: Efficiently Scaling through Space, Time, and Tasks”. In: *International Conference on Learning Representations*. 2024.

[221] Fan Zhang and Michael Gienger. “Affordance-based Robot Manipulation with Flow Matching”. In: *arXiv preprint arXiv:2409.01083* (2024).

[222] Tony Z Zhao et al. “Learning fine-grained bimanual manipulation with low-cost hardware”. In: *Proceedings of Robotics: Science and Systems*. 2023. DOI: 10.15607/RSS.2023.XIX.016. URL: https://www.roboticsproceedings.org/rss19/p016.html.

[223] Hongkuan Zhou et al. “Language-Conditioned Imitation Learning with Base Skill Priors under Unstructured Data”. In: *arXiv preprint arXiv:2305.19075* (2023).

[224] Hongyi Zhou et al. “BEAST: Efficient Tokenization of B-Splines Encoded Action Sequences for Imitation Learning”. In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025. URL: https://openreview.net/forum?id=rQCl1sf62w.

[225] Yanqi Zhou et al. *Mixture-of-Experts with Expert Choice Routing*. 2022. arXiv: 2202.09368 [cs.LG].

[226] Brianna Zitkovich et al. “RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control”. In: *Proceedings of The 7th Conference on Robot Learning*. Vol. 229. PMLR, 2023, pp. 2165–2183. URL: https://proceedings.mlr.press/v229/zitkovich23a.html.

A. Supplementary Material for BESO

A.1. BESO Hyperparameters

Model Type	BESO					Seq-BESO				
Hyperparameter	Block-Push	Relay-Kitchen	Aligning	Img-C	Lang-C	Block-Push	Relay-Kitchen	Img-C	Lang-C	Real World
Hidden Dimension	240	360	72	768	768	240	192	512	512	1024
Hidden Layers	4	6	4	6	6	4	6	6	6	8
Obs Seq Len	5	5	5	2	2	1	1	1	1	1
Act Seq Len	1	1	1	1	1	4	4	10	10	10
Goal window size	1	2	2	1	1	1	1	1	1	1
Number Heads	12	6	6	4	4	12	8	8	8	8
Attention Dropout	0.05	0.3	0.3	0.2	0.2	0.05	0.3	0.3	0.3	0.3
Residual Dropout	0.05	0.1	0.1	0.1	0.1	0.05	0.1	0.1	0.1	0.1
Learning rate	1e-4	1e-4	5e-4	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4
Optimizer	Adam	Adam	Adam	AdamW	AdamW	Adam	Adam	AdamW	AdamW	AdamW
σ_{max}	1	1	3	1	1	5	5	80	80	80
σ_{min}	0.05	0.005	0.005	0.005	0.005	0.05	0.005	0.001	0.001	0.001
σ_{data}	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Noise distribution	Log-Logistic	Log-Logistic	Log-Logistic	Log-Logistic	Log-Logistic	Log-Logistic	Log-Logistic	Log-Logistic	Log-Logistic	Log-Logistic
EMA	True	True	True	True	True	True	True	True	True	True
Vision Encoder	None	None	None	ResNet-18	FiLM-ResNet-18	None	None	ResNet-18	FiLM-ResNet-18	FiLM-ResNet-18
Sampler Type	DDIM	DDIM	DDIM	DDIM	DDIM	DDIM	DDIM	DDIM	DDIM	DDIM
Noise scheduler	Linear	Linear	Exp	Exp	Exp	Linear	Linear	Exp	Exp	Exp
Batch Size	1024	1024	1024	64	512	1024	1024	512	512	512
Train steps	60k	40k	40k	120k	30k	40k	40k	40k	30k	100k

Table A.1.: Overview of the most important Hyperparameters for BESO for tested environments. Img-C refers to image-only CALVIN experiments and Lang-C refers to Language-conditioned CALVIN benchmark experiments.

A summary of key hyperparameters of BESO is listed in Table A.1. We observe, that transformer specific-hyperparameters such as the dropout rates require tuning according to the task, while general diffusion hyperparameters remain consistent across different tasks.

Architecture Hyperparameters. For state-based experiments, we found that smaller transformer architectures with a hidden dimension of $180 - 320$ are typically enough to achieve very high performance across various tasks. Image-based experiments benefit from larger hidden dimensions ($512 - 1024$) with $6 - 8$ transformer layers. Dropout is crucial in image-based transformer models, with recommended rates of 0.3 for attention and 0.1 for other transformer components.

Noise Range and Normalization. BESO performs optimally when actions are diffused within $[-1, 1]$ with a noise range of $[0.005, 1]$ for single-action prediction. Seq-BESO uses a range of $[0.001, 80]$ for action chunk prediction in image-based settings. These values align with those used in image-generation Diffusion Models Karras et al. [83]. For the input, we recommend normalizing the data with a mean of 0 and a standard deviation of 1.

Training Noise distribution. During training, noise values are sampled from a predefined noise distribution $P(\sigma)$. While log-normal distribution is the standard in score-based diffusion literature [83], it has additional hyperparameters ($\sigma_{\text{std}}, \sigma_{\text{max}}$) that require environment specific tuning. Empirically we found that recommended values from prior work [83], optimized for image generation, are suboptimal for action diffusion. Thus, we use the truncated LogLogistic($\alpha = 0.5, \beta = \exp(0.5)$), which does not require additional parameters and works well in all settings. This distribution has a higher probability density on larger noise values.

Optimization. We use Adam for state-based environments and AdamW optimizer for image-based settings with a weight of 0.05 and $\beta = [0.9, 0.9]$. In all experiments we use a learning rate of 0.0001. BESO trains stable at standard learning rates. We also employ Exponential Moving Average (EMA) to optimize model weights with a decay rate of 0.999.

A.2. BESO Baselines

The MLP-based models have 4 layers with 512 neurons and use the ReLU activation function. All diffusion models have the same transformer backbone, and C-BeT uses its recommended parameters. During training, the Adam optimizer was used with a learning rate of 0.001 for MLP models and $1e-4$ for transformer models. The batch size for MLP models was 512, while it was 1024 for transformer models, except for BeT, which used a batch size of 64 as recommended in [31].

GCBC For the GCBC model, the goal is concatenated with the state and fed into the 4-layer MLP architecture with a dropout rate of 0.1. For the real-world experiments, we use a 8 layer GPT Transformer model with a 1024 latent dimension and FiLM-conditioned ResNets and CLIP language

embeddings. The real-world baseline also uses action chunking to predict 10 actions ahead for a fair comparison of the different action representations.

GC-IBC The GC-IBC model uses the same MLP architecture as GCBC and is optimized using the InfoNCE loss with additional energy-regularization and Wasserstein Gradient loss. During experiments, adding a penalty term with $\lambda = 0.005$ to restrict the average energy improved training stability [47]. Given the large number of tunable hyperparameters for IBC, we ran a hyperparameter search to determine the best ones. We want to note, that the model results of EBM were sensitive to initial seeds and it is hard to get consistent results. Similar observations of IBC performance have been reported in related work [15, 27].

C-BeT For the performance of C-BeT, we use the recommended parameters from Cui et al. [31] for all tested environments from their paper. Our reported results are marginally worse than the ones reported in the original work since they do not average it over 10 seeds. For the image-based experiments on CALVIN, we use the same hyperparameters as noted for BESO to make a fair comparison.

Latent Motor Plans and HULC The LMP model was evaluated on the Kitchen and Block Push environments with extensive hyper-parameter sweeps to find the best-performing configuration. We used the improved LMP variant, HULC [129], which uses a different KL-divergence weighting term and a transformer model the Seq2Seq CVAE. On the CALVIN environment, the proposed parameters from prior work were used [167] and for the language-conditioned variant we directly compare against HULC [129] and report the values from their paper to guarantee a comparison against the best performing model.

RIL For the low-level policy of kitchen and block push we use 4 layers with 512 neurons each. For the CALVIN task, we used the baseline version from [167] and kept the hyperparameters the same for training.

Diffusion-X The baseline from [147] uses the same hyperparameters of our transformer model reported in Table A.1 to guarantee a fair comparison. Diffusion-X uses 50 inference steps on the kitchen task combined with an additional 10 fine-tuning steps at the lowest noise level, while we use 20 inference steps for the block-push environment and additional 8 fine-tuning steps. Further, it applies X-additional denoising steps at the lowest noise level.

A.3. Additional Experiments

A.3.1. BESO Rollout Examples and Failure Cases

Environment. Our robot setup features a toy kitchen with a microwave, oven, cooler, and sink. Additional objects include a banana, pot, and toaster with toast for the robot to interact with. BESO evaluates a 10-task subset of the 20-task kitchen repertoire used for FLOWER, using the same play-data collection. Example BESO rollouts are visualized in Figure A.1 and Figure 3.6.

Failure Cases. Both policies successfully grounded all language instructions to the correct objects and consistently moved near relevant objects, demonstrating good task comprehension. However, they exhibited different types of failures in execution: BESO generally performed well across most tasks, showing robust behavior. Its main challenge was with toaster-related tasks. When BESO missed the toaster lever or failed to pick up the toast, it would get stuck and not attempt to retry the action. This highlights a potential area for improvement in BESO's error recovery capabilities. GCBC, while able to ground tasks well, struggled more with action execution. It often displayed indecisiveness, such as hovering over the banana without picking it up, or picking up the pot and holding it for extended periods (up to 20 seconds) without completing the intended action. GCBC also showed issues with precision, frequently missing oven and microwave handles and the oven tray by small margins. It also had difficulty accurately locating and interacting with objects like the banana, often attempting to pick it up from incorrect positions.

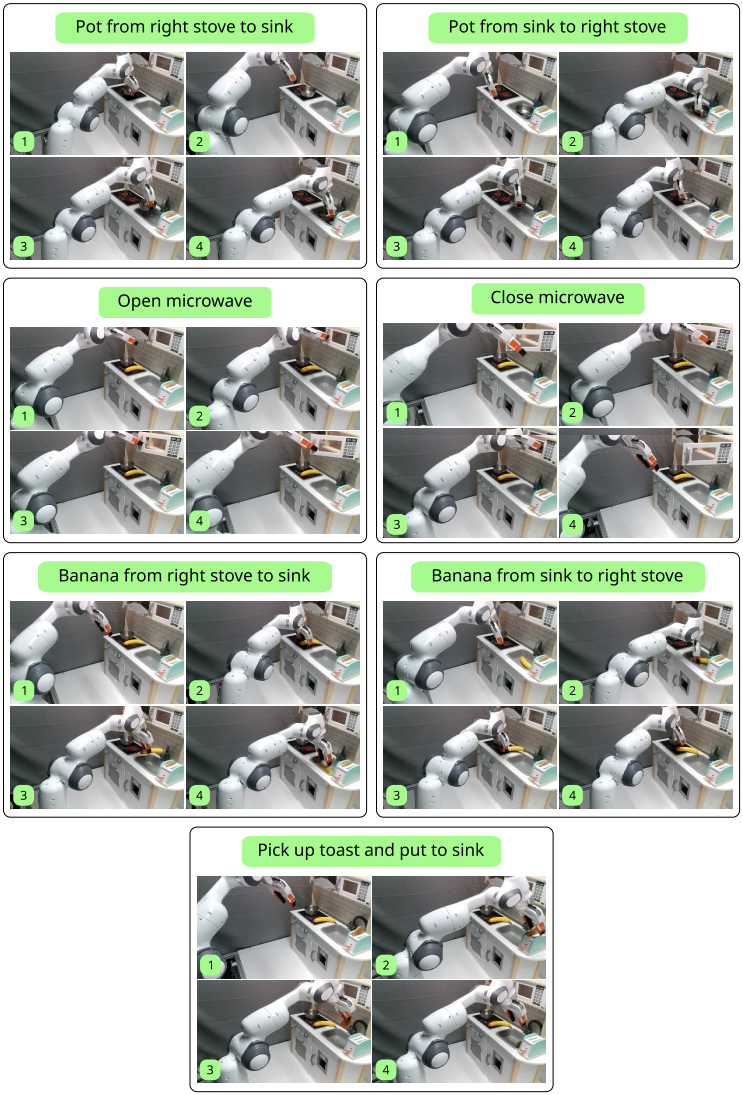


Figure A.1.: Visualization of BESO Rollouts in the Play Kitchen. The sequences showcase successful policy rollouts from BESO for 7 tasks in a real-world play kitchen environment, demonstrating the model's effectiveness across diverse kitchen-related activities.

B. Supplementary Material for MDT

B.1. MDT Hyperparameters

We summarize all core hyperparameters for training MDT in Table B.1.

B.2. MDT Baselines

MT-ACT. An overview of the used hyperparameters of MT-ACT is given in Table B.1. We tried to stay close to the recommended hyperparameters from the original paper [9] but optimized the action prediction length and the Kl-β factor for CALVIN. Empirically, we found that FiLM-conditioned ResNets do not perform well conditioned on image-goal or in combination with not using any FiLM conditioning when having image goals. Thus, we adopted default ResNets as the vision encoders for MT-ACT, as other experiments with pre-trained Voltron-Encoders [82] did not show good results and reduced the performance over 20% on the CALVIN benchmark ABCD→D.

Distill-D. We use the reported hyperparameters of Chi et al. [27] in combination with two ResNets18 and frozen CLIP encoders for visual and language goals as described in Ha, Florence, and Song [58]. The resulting model contains 296.6 million trainable parameters in the 1D-CNN and an additional 22.4 million parameters for two ResNets-18. We also experimented with pretrained Voltron embeddings for Distill-D, however similar to MT-ACT Voltron did not show any performance improvements.

Hyperparameter	MT-ACT	MDT-V	MDT
Number of Encoder Layers	4	4	4
Number of Decoder Layers	6	4	6
Attention Heads	8	8	8
Action Chunk Size	10	10	10
Goal Window Sampling Size	49	49	49
Hidden Dimension	512	384	512
Action Encoder Layers	2	-	-
Action Encoder Hidden Dim	192	-	-
Latent z dim	32	-	-
Image Encoder	ResNet18	Voltron V-Cond	ResNet18
Attention Dropout	0.1	0.3	0.3
Residual Dropout	0.1	0.1	0.1
MLP Dropout	0.1	0.05	0.05
Input Dropout	0.0	0.0	0.0
Optimizer	AdamW	AdamW	AdamW
Betas	[0.9, 0.9]	[0.9, 0.9]	[0.9, 0.9]
Transformer Weight Decay	0.05	0.05	0.05
Other weight decay	0.05	0.05	0.05
Batch Size	512	512	512
Trainable Parameters	122 M	40.0 M	75.1 M
σ_{max}	-	80	80
σ_{min}	-	0.001	0.001
σ_t	-	0.5	0.5
Time steps	-	Exponential	Exponential
Sampler	-	DDIM	DDIM
Kl-β	50	-	-
Contrastive Projection	-	MAP	Single Token 1
Goal Image Encoder	CLIP ViT-B/16	CLIP ViT-B/16	CLIP ViT-B/16
Goal Lang Encoder	CLIP ViT-B/32	CLIP ViT-B/32	CLIP ViT-B/32

Table B.1.: Summary of all the Hyperparameters for the MDT policy used in the CALVIN experiments and the ones of MT-ACT.

B.3. MDT Additional Ablations

We experimented with different sampling strategies and ranges of future windows. We found that geometric sampling with a distribution probability of $p = 0.1$ works well in all tested settings on CALVIN. Other experiments with random sampling showed small drops in performance, while trials with key-state-based goal states similar to RLBench [75] did not work well in any

Hyperparameter	CALVIN	LIBERO	Real World
Number of Layers	6	2	2
Hidden Dimension	192	192	192
Image resolution	112	112	112
Masking Ratio	0.75	0.75	0.75
MLP Ration	4	4	4
Patch size	16	16	16
Norm Pixel Loss	True	True	True

Table B.2.: Overview of the chosen hyperparameters for our Image Demasking Model used in the MGF loss, that uses a vision transformer architecture.

Hyperparameter	Distill-D
Action Chunk Size	8
Timestep-embed Dimensions	256
Image Encoder	ResNet18
Channel Dimensions	[512, 1024, 2048]
Learning Rate	1e-4
$\sigma_{\max}$	80
$\sigma_{\min}$	0.001
σ_t	0.5
Time steps	Exponential
Sampler	DDIM
Sampling Steps	10
Trainable Parameters	318 M
Optimizer	AdamW
Betas	[0.9, 0.9]
Goal Image Encoder	CLIP ViT-B/16
Goal Lang Encoder	CLIP ViT-B/32

Table B.3.: Overview of the hyperparameters for Distill-D on the CALVIN and LIBERO benchmark. Our code is based on the Diffusion-policy implementation from Chi et al. [27] with our continuous-time diffusion variant. To guarantee a fair comparison the hyperparameters for Distill-D and MDT regarding diffusion are the same.

	CALVIN ABCD	LIBERO-Spatial
1	4.50 ± 0.02	64.4 ± 0.4
3	**4.52 ± 0.02**	**67.5 ± 0.2**
9	4.44 ± 0.03	65.6 ± 0.5

Table B.4.: Ablation on the best prediction horizon for Masked Generative Foresight. Results are reported on CALVIN ABCD→D with MDT-V and LIBERO-Spatial with MDT.

Policy	Avg. Len. CALVIN
MT-ACT	2.80 ± 0.03
MT-ACT + $\mathcal{L}_{\text{MGF}}$	**4.03 ± 0.08**

Table B.5.: Evaluation of the Performance Increase of the MT-ACT policy with the additional Masked Generative Foresight Loss on the CALVIN ABCD→D challenge.

setting. Thus, we decided to use the same strategy for CALVIN and real robot experiments.

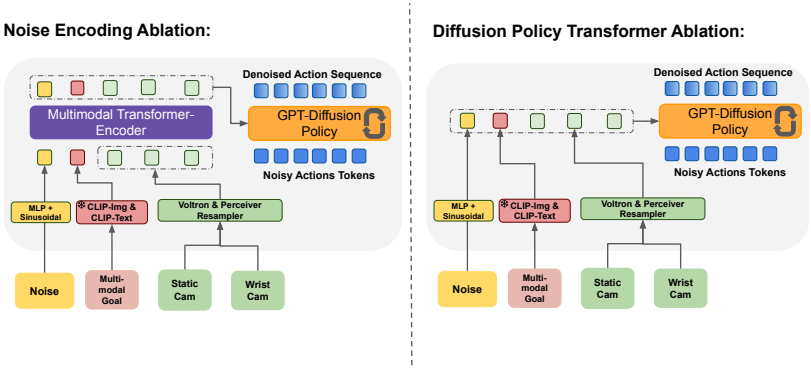


Figure B.1.: Overview of two Diffusion Transformer Baseline Architectures for MDT. The first variant uses a transformer encoder but also processes the noise as a token. The second one is the Transformer Diffusion policy presented in Chi et al. [27] without any encoder.

Method	$\mathcal{L}_{\text{CLA}}$	$\mathcal{L}_{\text{MGF}}$	No. Instructions in a Row (1000 chains)					
			1	2	3	4	5	**Avg. Len.**
MDT-V Abl. 1	×	×	0.914	0.782	0.675	0.588	0.487	3.58 ± 0.18
MDT-V Abl. 2	×	×	0.693	0.405	0.190	0.092	0.031	1.41 ± 0.04
MDT-V	×	×	0.971	0.907	0.840	0.766	0.698	4.18 ± 0.10
MDT-V	✓	×	0.977	0.927	0.868	0.808	0.786	4.32 ± 0.06
MDT-V	×	✓	0.986	0.946	0.903	0.851	0.794	4.48 ± 0.03
MDT-V	✓	✓	0.989	0.958	0.916	0.862	0.801	4.52 ± 0.02
MDT	×	×	0.882	0.753	0.653	0.557	0.481	3.34 ± 0.06
MDT+Goal ResNets	×	×	0.886	0.764	0.651	0.565	0.48	3.34 ± 0.05
MDT	✓	✓	0.978	0.938	0.888	0.831	0.77	4.41 ± 0.03
MDT+Goal ResNets	✓	✓	0.978	0.923	0.862	0.807	0.735	4.31 ± 0.08

Table B.6.: Overview of the performance influence of MGF and Contrastive Alignment on MDT-V on the CALVIN ABCD→D challenge. In addition, the performance of both transformer ablations are also shown. Moreover, results for MDT with ResNets as image goal encoders are reported with and without auxiliary objectives. The results are reported over 1000 rollouts averaged over 3 seeds. Numerical entries, including reported averages and uncertainties, are reproduced from Table IX of the final RSS publication [163]; they have not been recomputed from the displayed chain probabilities.

Masking Rate	CALVIN D	LIBERO-Spatial
0.5	3.7 ± 0.04	67.8 ± 0.3
0.75	3.72 ± 0.05	67.5 ± 0.2
1	3.68 ± 0.03	63.7 ± 0.3

Table B.7.: Ablation on different Masking Rates for Masked Generative Foresight. Results are reported on CALVIN D→D with MDT-V and on LIBERO-Spatial with MDT.

B.3.1. MDT Example Rollouts

B.3.2. MDT Real Robot Results

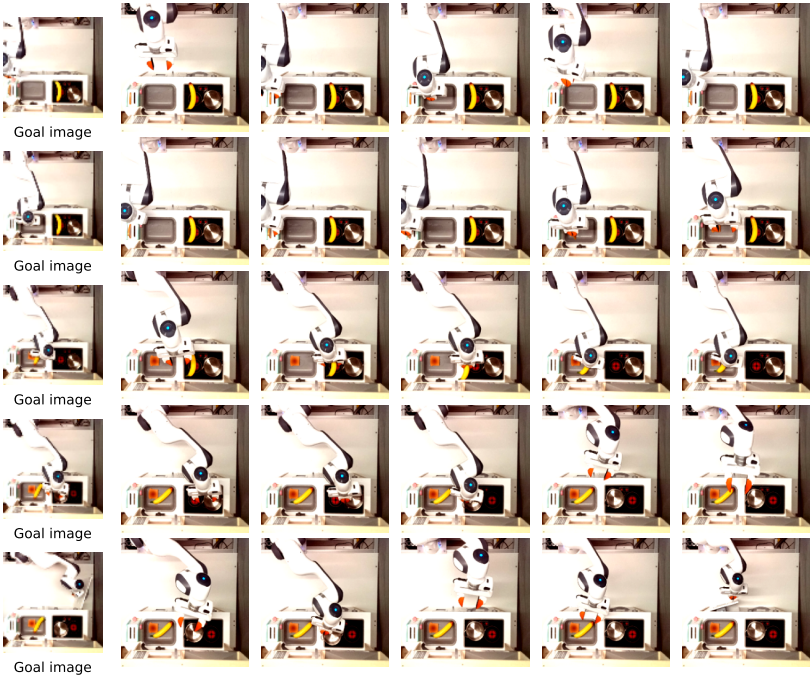


Figure B.2.: Real Robot rollouts with goal image conditioning using MDT. The first column shows the goal image used for the rollout. 4 out of 6 tasks are successful. The robot fails to open the oven door and opens the ice box instead.



Figure B.3.: Overview of the real robot kitchens setup. The left image shows the play kitchen with all its objects, while the right image shows the cameras and second robot used for data collection with our robot used for teleoperation.

Task	No.	MT-ACT	MDT	MDT MGF
Banana from rt stove to sink	1	0	0.8	1.0
Banana from sink to rt stove	2	0	0.8	1.0
Pot from rt stove to sink	3	0	0	0
Pot from sink to rt stove	4	0.4	0.8	0
Pot from lt stove to sink	5	0	0.8	0.6
Pot from sink to lt stove	6	0	0	1.0
Pot from lt to rt stove	7	0.2	0.2	0
Pot from rt to lt stove	8	0	0.6	1.0
Open microwave	9	1.0	1.0	1.0
Close microwave	10	0	1.0	0
Open oven	11	1.0	1.0	1.0
Close oven	12	0	0	0
Open ice box	13	1.0	1.0	1.0
Close ice box	14	0	1.0	0
Pull oven tray	15	0.6	0	0.2
Push oven tray	16	0.4	0	0.2
Banana from rt stove to tray	17	0	0	0.4
Banana from tray to rt stove	18	0	0	1.0
Push toaster	19	0	0.2	1.0
Toast to sink	20	0.4	1.0	1.0

Table B.8.: Detailed results of our real robot kitchen experiments for single-task setting with language goals of MDT. All results are averaged over 5 rollouts. "right" and "left" are abbreviated with "rt" and "lt".

MT-ACT (Language Goals)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
1: Random	0.0	0.0	0.0	0.0	0.0	0.0
2: Open Close All	1.0	0.75	0.5	0.0	0.0	0.0
3: Stovetop Sink	0.5	0.0	0.0	0.0	0.0	0.0
4: Oven	0.5	0.0	0.0	0.0	0.0	-
MT-ACT (Image Goals)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
1: Random	0.0	0.0	0.0	0.0	0.0	0.0
2: Open Close All	0.0	0.0	0.0	0.0	0.0	0.0
3: Stovetop Sink	0.5	0.0	0.0	0.0	0.0	0.0
4: Oven	0.0	0.0	0.0	0.0	0.0	-
MDT (Language Goals)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
1: Random	0.75	0.75	0.0	0.0	0.0	0.0
2: Open Close All	1.0	1.0	0.75	0.0	0.0	0.0
3: Stovetop Sink	0.75	0.0	0.0	0.0	0.0	0.0
4: Oven	0.5	0.0	0.0	0.0	0.0	-
MDT (Image Goals)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
1: Random	0.5	0.5	0.25	0.25	0.0	0.0
2: Open Close All	0.0	0.0	0.0	0.0	0.0	0.0
3: Stovetop Sink	0.0	0.0	0.0	0.0	0.0	0.0
4: Oven	0.0	0.0	0.0	0.0	0.0	-
MDT + $\mathcal{L}_{\text{MGF}}$ + $\mathcal{L}_{\text{CLA}}$ (Language Goals)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
1. Random	1.0	0.75	0.25	0.25	0.0	0.0
2. Open Close All	1.0	1.0	0.25	0.0	0.0	0.0
3. Stovetop Sink	0.5	0.25	0.0	0.0	0.0	0.0
4. Oven	0.75	0.25	0.0	0.0	0.0	-
MDT + $\mathcal{L}_{\text{MGF}}$ + $\mathcal{L}_{\text{CLA}}$ (Image Goals)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
1: Random	0.25	0.25	0.25	0.25	0.0	0.0
2: Open Close All	0.5	0.0	0.0	0.0	0.0	0.0
3: Stovetop Sink	0.25	0.0	0.0	0.0	0.0	0.0
4: Oven	0.5	0.0	0.0	0.0	0.0	-

Table B.9.: Detailed results of our long-horizon, real robot multi-task experiments for MDT and baselines.

C. Supplementary Material for MoDE

C.1. MoDE Hyperparameters

Hyperparameter	CALVIN ABCD	CALVIN ABC	LIBERO-10	LIBERO-90	Pret-MoDE
Number of Transformer Layers	8	8	8	8	12
Number Experts	4	4	4	4	4
Attention Heads	8	8	8	8	8
Action Chunk Size	10	10	10	10	10
History Length	1	1	1	1	1
Embedding Dimension	1024	1024	1024	1024	1024
Image Encoder	FiLM-ResNet18	FiLM-ResNet50	FiLM-ResNet18	FiLM-ResNet18	FiLM-ResNet50
Goal Lang Encoder	CLIP ViT-B/32	CLIP ViT-B/32	CLIP ViT-B/32	CLIP ViT-B/32	CLIP ViT-B/32
Attention Dropout	0.3	0.3	0.3	0.3	0.3
Residual Dropout	0.1	0.1	0.1	0.1	0.1
MLP Dropout	0.1	0.1	0.1	0.1	0.1
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW
Betas	[0.9, 0.95]	[0.9, 0.95]	[0.9, 0.95]	[0.9, 0.95]	[0.9, 0.95]
Learning Rate	1e-4	1e-4	1e-4	1e-4	1e-4
Transformer Weight Decay	0.05	0.05	0.05	0.05	0.1
Other weight decay	0.05	0.05	0.05	0.05	0.1
Batch Size	512	512	512	512	512
Train Steps in Thousands	30	25	15	30	300
σ_{max}	80	80	80	80	80
σ_{min}	0.001	0.001	0.001	0.001	0.001
σ_t	0.5	0.5	0.5	0.5	0.5
EMA	True	True	True	True	True
Time steps	Exponential	Exponential	Exponential	Exponential	Exponential
Sampler	DDIM	DDIM	DDIM	DDIM	DDIM
Parameter Count (Millions)	460	460	460	460	685

Table C.1.: Summary of all the Hyperparameters for the MoDE policy used in our experiments.

C.2. MoDE Baselines

Below we explain several baselines used in the experiments in detail:

Diffusion Policy-CNN/T Inspired by [27], we evaluate extension of the DDPM based Diffusion Policy framework for goal-conditioned Multi-task

learning. We evaluate two versions: the CNN-based variant and the Diffusion-Transformer variant, that is conditioned on context and noise using cross-attention. For our experiments we also use EDM-based Diffusion framework for fair comparison against MoDE. We optimized the ideal number of layers and latent dimension for the Transformer baseline and our final version uses 8 layers with a latent dimension of 1024. Larger or smaller variants resulted in lower average performance.

RoboFlamingo. RoboFlamingo [105] is a VLM finetuned for behavior generation. The authors use a 3 billion parameter Flamingo model [2] and fine-tune it on CALVIN by freezing the forward blocks and only fine-tuning a new Perceiver Resampler module to extract features from a frozen vision-transformer image encoder and the cross-attention layers to process the image features. Finally, a new action head is learned to generate actions. Overall, the finetuning requires training approx. 1 billion of the parameters. We report the reported results from the paper since they use the standard CALVIN evaluation suite.

SuSIE. This model first finetunes InstructPix2Pix, an image-generation diffusion model, that generates images conditioned on another image and a text description [19] on the local CALVIN robotics domain and uses it as a high-level goal generator. The low-level policy is a CNN-based Diffusion Policy, that predicts the next 4 actions given the current state embedding and desired sub-goal from the high-level policy [12].

GR-1 A causal GPT-Transformer model [208], that has been pretrained on large-scale generative video prediction of human videos. Later, the model is finetuned using co-training of action prediction and video prediction on CALVIN. We report the results directly from their paper for the CALVIN benchmark.

C.3. MoDE Mixture-of-Experts Ablations

Only the CALVIN ABC ablation runs use a separate, weaker training run than the main ABC benchmark in Chapter 5. Their relative performance is compared within the corresponding ablation setting.

Q: How does the Load Balancing Loss influence the Expert Distribution?

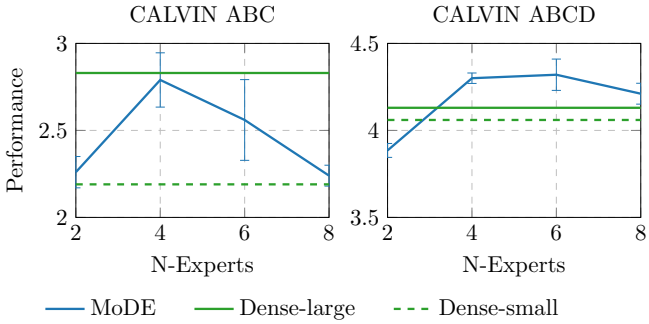


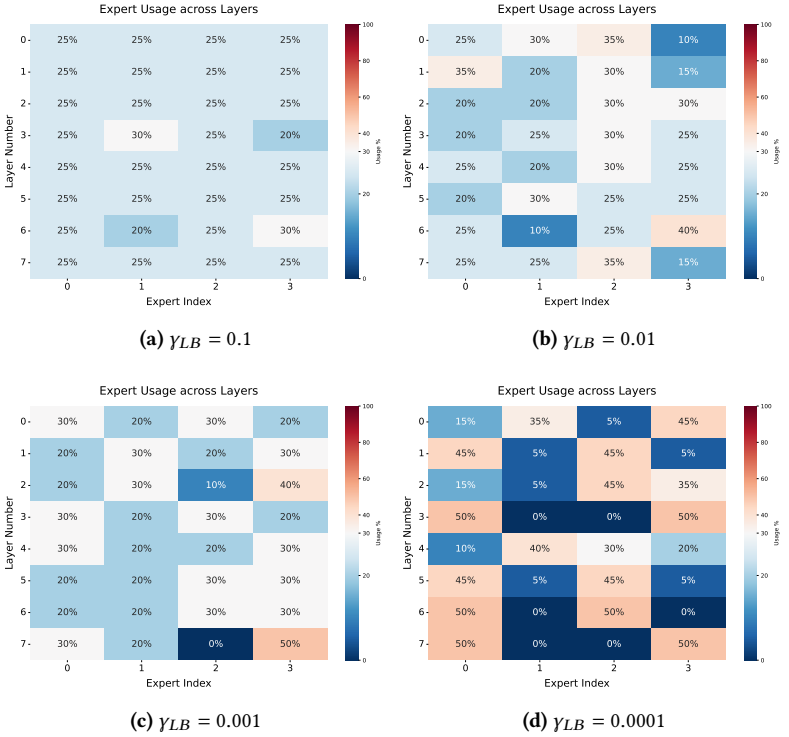
Figure C.1.: Scaling performance of MoDE and Dense-MoDE on CALVIN ABC and ABCD environments. The plots show average performance for 2 to 8 experts using best-performing variants for each environment.

We analyze how the load balancing loss affects expert distribution across noise levels by training MoDE on LIBERO-10 with varying load balancing weights $\gamma_{LB} \in [0.1, 0.01, 0.001, 0.0001]$ Figure C.2 visualizes the resulting expert distributions.

With a high load balancing loss of 0.1, experts are used almost uniformly across all layers with minor variations in two out of eight layers (Figure C.2a). However, this enforced uniformity comes at a cost - the average performance drops to 0.9. This result suggests that enforcing equal expert usage across noise levels may constrain the model's learning capacity.

At 0.01, we observe a more flexible distribution while maintaining good overall expert utilization (Figure C.2b). Within individual layers, expert usage varies from 10% to 40% in various layers. This enables for specialization while preventing any expert from becoming dominant. This configuration achieves the best average performance and provides evidence that moderate specialization benefits the policy most.

As we reduce γ_{LB} further to a range of $[0.001, 0.0001]$, the expert distribution gets worse and the model there is an expert collapse is happening for lb of 0.0001 in several layers (Figure C.2c and Figure C.2d). These values show, that a the load balancing weight is crucial to guarantee a good usage of all experts for the model. The lower performance of these models (0.86 for $\gamma_{LB} = 0.001$



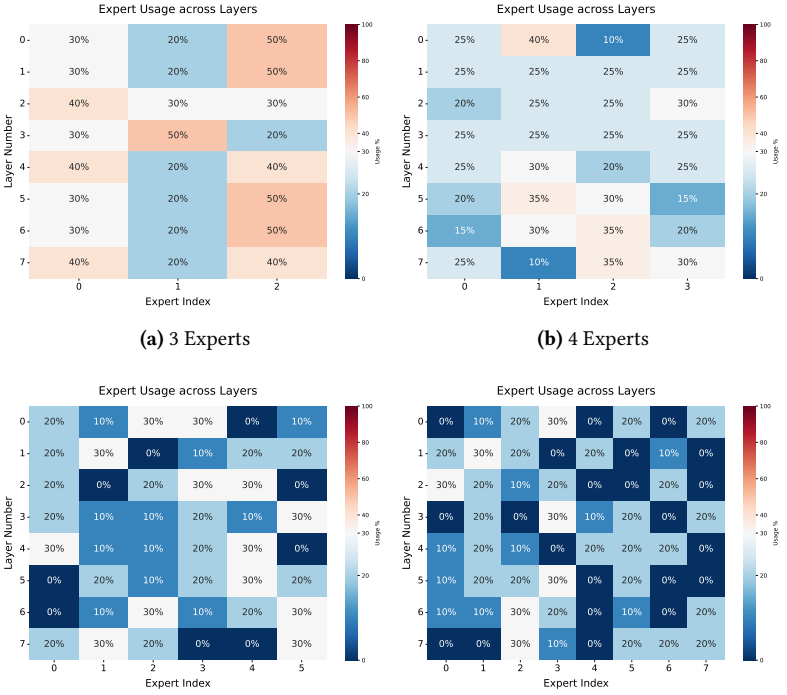
(a) $\gamma_{LB} = 0.1$ **(b)** $\gamma_{LB} = 0.01$

(c) $\gamma_{LB} = 0.001$ **(d)** $\gamma_{LB} = 0.0001$

Figure C.2.: Average Expert Utilization for different Load Balancing Weights across all denoising levels for MoDE.

and 0.83 for $\gamma_{LB} = 0.0001$) suggests that balanced expert participation is important for optimal Diffusion Policy performance.

Q: What happens with increased number of Experts? Next, we study the load balancing distribution for models that use more experts. Our experiments with 6 and 8 expert variants on LIBERO-10 reveal challenges in scaling beyond 4 experts. As shown in Figure C.3, configurations with more experts consistently underutilize their capacity, with most layers actively using only 4 experts even with increased load balancing regularization ($\gamma = 0.1$). These larger configurations also show decreased performance, indicating that 4 experts provides an optimal balance between representation capacity and learning efficiency. We hypothesize that additional experts impair the



(a) 3 Experts

(b) 4 Experts

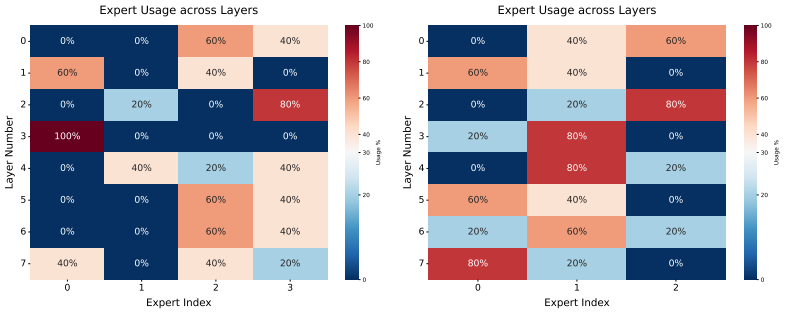
(c) 6 Experts

(d) 8 Experts

Figure C.3.: Average Expert Utilization for different number of used experts across all denoising levels.

model's ability to learn effective representations from image and language inputs while maintaining noise-level specialization. This phenomenon persists even when increasing the load balancing loss factor, further supporting our finding that 4 experts offers the best trade-off between model capacity and performance.

Q: How is the expert distribution with topk=1? We further study the impact of using $top_k = 1$ for MoDE. We analyze MoDE's behavior with topk=1 routing, both with and without a shared expert, using 4 experts total. As visualized in Figure C.4, topk=1 configurations struggle to maintain balanced expert utilization, typically collapsing to using only two of the four available experts. While this configuration shows only modest performance degrada-



(a) topk=1 **(b)** topk=1 with shared Expert

Figure C.4.: Average Expert Utilization for two variants that use topk=1. The left plot shows MoDE with 4 experts, the right shows MoDE with 4 experts, where 1 is used in all settings as a shared one. Overall, MoDE struggles to equally distribute tokens across experts in this setting.

tion on LIBERO-10, its impact is more visible on the challenging CALVIN ABC benchmark, where the average rollout length drops from 3.34 (topk=2) to 2.72 (topk=1). These CALVIN ABC scores come from a separate, weaker training run used for the top-k ablation. The relative change quantifies the routing choice in that ABC setting; the main non-pretrained ABC benchmark reports 3.39 (Chapter 5). These results highlight the importance of expert interpolation across noise levels, with topk=2 providing the necessary flexibility to blend expert specializations for optimal performance.

Q: How do experts specialize across noise levels?

Finally, we analyze the different experts loads for various noise levels of our biggest model after pretraining on diverse robotics data. Detailed results are shown in Figure C.5.

Our analysis of the expert distribution reveals several key insights about how the model learns to organize its mixture-of-experts architecture:

- **Noise-Level Specialization**
  - Experts specialize between high-noise (σ_1-σ_7) and low-noise (σ_8-σ_{10}) regimes, particularly visible in L4 where E3 and E0 show distinct high activation patterns in different noise regimes

  - Clear transition point around σ_8 where expert utilization patterns shift, most evident in L4 and L5 with stark changes in expert activation
  - Smooth handoff between denoising phases, exemplified in L7 where expert activations gradually transition across noise levels

- **Layer-wise Organization**
  - Early layers (L0-L4) show distinct expert specialization, particularly visible in L4's strong alternating expert patterns
  - Middle layers (L5-L7) demonstrate more distributed expert utilization, shown by more balanced activation patterns
  - Later layers (L8-L11) return to clearer specialization, evident in L11's distinct expert preferences
  - First layer (L0) shows particularly strong specialization for low-noise scenarios, with clear expert preferences in the final denoising steps

- **Expert Role Distribution**
  - Different experts develop specialized roles, most clearly visible in L4 where experts show strong preferences for specific noise ranges
  - Some experts consistently handle high-noise denoising (evident in L3 and L4) while others focus on low-noise refinement (for example in L7-L8)
  - "Transitional experts" emerge in middle noise ranges, particularly visible in L5's balanced activation patterns
  - Model employs different expert combinations across layers, well demonstrated in the contrasting patterns between L4 and L8

- **Load Balancing Properties**
  - Effective load balancing achieved across experts, particularly visible in the distributed patterns of L5-L7
  - No single expert dominates across all noise levels, as shown by the varied activation patterns in each layer

– Smooth transitions between noise regimes, most evident in the gradual activation changes in L4 and L5

These findings provide strong evidence that MoDE effectively partitions the denoising process across its experts, with each expert naturally specializing in different phases. The emergence of this organization without explicit supervision supports the effectiveness of our architectural choices and routing strategy.

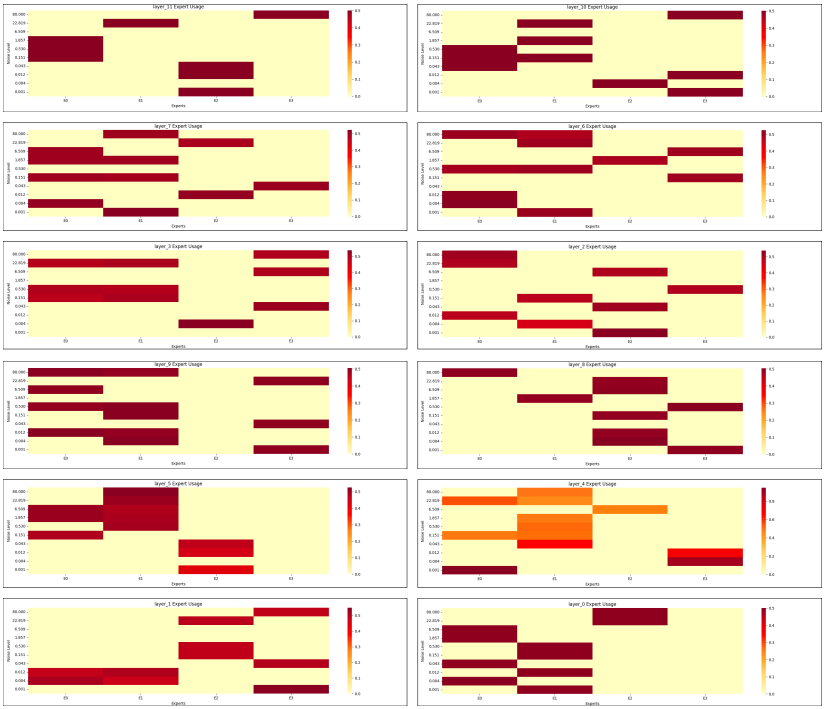


Figure C.5.: Visualization of used experts in pretrained MoDE. This image shows the expert usage distribution across 12 layers in a Mixture of Experts (MoE) model. Each subplot represents a different layer (from layer 0 to layer 11), with experts labeled E0, E1, E2, and E3 on the x-axis. The y-axis indicates the log-scaled token count, displaying the frequency of tokens routed to each expert within that layer. The color gradient indicates the proportion of tokens assigned to each expert, where darker colors represent higher usage.

D. Supplementary Material for FLOWER

D.1. FLOWER Hyperparameters

D.2. FLOWER Model and Pretraining Details

We pretrain FLOWER on different datasets mixes that are described in Table D.4 using one cluster node with 4 H100 GPUs for 48 hours. We pretrain all variants using a single static image and only use proprioception signal for the bimanual settings. We set the action chunk length to 20 across all settings and condition the model on single image from the current state only for maximum efficiency. Training is conducted using HuggingFace Accelerate for optimized Multi-GPU Training. We created custom PyTorch wrapper for the OXE Torch Dataloaders [30] inspired by efforts from OpenVLA [89]. We

	SIMPLER	CALVIN	LIBERO	Aloha	Real World Kitchen
Action Space Encoders	2-layered MLP				
Action Space Decoders	Linear Attention				
Number of Flow-T Layers	18				
Latent Dimension	1024				
Number of Heads	16				
Position Embedding	1D Rope				
Sampling Distribution	Uniform				
Attention Dropout	0.1				
MLP Dropout	0.1				
Residual Dropout	0.1				
Act Seq Length	10	10	10	100	20
Denoising Steps	4	4	4	8	5
Multistep	5	10	10	2: Insert 100:Transfer	15
Camera Views	[Primary]	[Primary, Wrist]	[Primary, Wrist]	[Primary]	[Primary, Secondary]
Use Proprio	False	False	False	True	False
Action Space	Delta EEF	Delta EEF	Delta EEF	Bi-Joint	Joint
Frequency	3/5	10	10	50	6

Table D.1.: Overview of Hyperparameters used for FLOWER across different Benchmarks

Name	Number of Parameters
ViT	360M
VLM	205M
Action Encoders	3.2M
Action Heads	31.8K
Global-AdaLN	28.3M
Cond Linear Proj.	1.0M
Timestep Embedder	1.3M
Cond Norm	1.0K
FreqEmbedder	1.3M
Flow Transformer	339M
Total Parameters FLOWER	947M

Table D.2.: Overview of Parameter Distribution across all Model Components of FLOWER.

train FLOWER using BF16 precision for optimized memory performance. For the Cross-Action-and Delta-EEF Mix we set the batch size to 256 and use 4 gradient accumulate steps to achieve an effective batch size of 1024. In total we trained for 300k-400k training steps depending on the dataset composition. We did not notice major training instabilities.

D.2.1. Pretraining Ablation Experiments

We tested several ideas, that did not work well in our maximum efficiency pretraining settings:

- **Using variable Length Action Chunks:** We experimented with flexible action chunks during training as done in CrossFormer [36]. However, we noticed slow convergence and lower performance for our SIMPLER experiments. Thus, we decided to use a constant chunk length for all datasets of 20. While many delta EEF datasets like Fractal operate on lower frequency and Aloha Setups like Biplay [32] require 50 Hz, we find 20 to be a good trade-off that enables easy finetuning to different lengths, thanks to our 1D Rope Position Embeddings.
- **Using Multiple Images with Custom Masking:** We also tested variations in pretraining that involve flexible image padding up to 3

Hyperparameter	Value
GPU Type	H100
N GPUS	4
Batch Size	256
Grad accumulate Steps	4
Optimizer FlowT	AdamW
LR Max FlowT	1e-4
LR Scheduler	warm-up with constant + cosine decay
Min LR FlowT	1e-5
Final LR FlowT	1e-5
FlowT LR scheduler phases	[0.01, 0.39, 0.6]
Max Desired Train Steps	600,000
Optimizer VLM	AdamW
Lr Max VLM	1e-5
Lr Scheduler	warm-up with constant + cosine decay
Min LR VLM	1e-7
Final LR VLM	1e-6
VLM LR scheduler phases	[0,1, 0.3, 0.6]
EMA	False
Weight Decay FlowT	0.1
Weight Decay VLM	0.001
Total Training Time	48 hours
Training Steps reached	350,000

Table D.3.: Hyperparameters for Pretraining FLOWER on all Pretraining Data Mixtures

Table D.4.: Dataset Distribution for Pretraining of FLOWER. Results are reported by percentage

Dataset	Percentage (%)
bridge_dataset	28.62
fractal_data	24.68
DROID	23.50
cmu_play_fusion	6.15
dobbe	5.94
libero_10_no_noops	4.41
libero_goal_no_noops	4.07
real_kitchen_lang	2.64

different views depending on the dataset. However, this reduces the overall training speed given the higher number of image tokens and required GPU memory. Also given the more complex setting, we found that the training time of 2 days not enough to achieve convergence. Thus, we limited pretraining to single image. However, our finetuning experiments show the flexibility of FLOWER to adapt to more image views without issues.

- **Mixture-of-Experts Approaches for the Flow Transformer**: We conducted architecture ablation experiments with more action-specialist components. FLOWER ablations used action-type specific MLPs with action-type specific LayerNorms. We also experimented with a shared additional MLP, that all action types use while combining the output with action-specific specialist MLP. However, the training was more memory intensive and had issues with NaN losses. It also showed slower convergence. We found the action-specific Global-AdaLN with all other parameters shared inside the Flow Transformer to work best. However, we believe that future research can address this issue to develop even more efficient architectures for cross-embodiment learning.

D.2.2. Language Prompt for the VLA

We encode the meta-information as part of our prompt for the VLM: *"Agent Type: [robot_type], Action Space: [action_space], Task: [task_description]".*

D.2.3. Custom Learning Rate Scheduler

In early experiments with an older version of our model, we observed a substantial drop in performance when training without our custom dual–optimizer learning rate scheduler (4.44 vs 0.8). Upon further investigation after finalizing the model, we found that this was largely due to instabilities in an earlier model variant rather than the absence of the scheduler itself. With our final architecture, replacing the custom scheduler with a standard constant learning rate of 2×10^{-5} leads to only a slight reduction in performance, confirming that the scheduler provides only minor benefits rather than being a critical requirement for the performance.

D.2.4. Details for Cross-Action Space Flow Transformer

Large transformer models often face stability challenges when simultaneously dealing with different data frequencies and distributions. We address this by allocating individual dual-RMSNorm parameters for each action type, capturing action-specific activation statistics more effectively than a single normalization.

Additionally, we replace standard feed-forward layers with SwiGLU blocks [176], a sandwich-style MLP that we express as follows. For an input vector $\mathbf{h}$,

$$\mathbf{y} \;=\; \Big(\mathrm{Norm}(\mathbf{W}_1\,\mathbf{h})\Big) \;\odot\; \mathrm{SiLU}\Big(\mathrm{Norm}(\mathbf{V}_1\,\mathbf{h})\Big), \tag{D.1}$$

where $\mathrm{SiLU}(x) = x\,\sigma(x)$, $\mathbf{V}$ and $\mathbf{W}$ are two linear matrices and $\mathrm{Norm}(\cdot)$ indicates RMSNorm in our implementation. Unlike LayerNorm, which subtracts the mean, RMSNorm normalizes each sample by its root-mean-square:

$$\mathrm{RMSNorm}(\mathbf{x}) \;=\; \frac{\mathbf{x}}{\sqrt{\frac{1}{d}\sum_{j=1}^{d} x_j^2 \;+\; \epsilon}}, \tag{D.2}$$

yielding smoother gradients and reduced training instability [219]. Furthermore, we apply query–key normalization [64] in both self- and cross-attention modules to mitigate large softmax outputs. This additional normalization has been used in many large-scale diffusion and flow transformer architectures in image generation [42] or Diffusion Policies [113].

Together, (i) extended RoPE embeddings, (ii) action-specific encoders and decoders, (iii) a shared AdaLN controller yielding action-specific normalization signals, and (iv) RMSNorm with SwiGLU MLPs enable our Flow Transformer to train stably across heterogeneous data and multiple action spaces. Its modular architecture requires minimal changes when adding new action types, retaining efficient scalability for a wide range of robotic tasks.

D.3. FLOWER Detailed Experiments

In this section we provide detailed results and additional comparisons for all simulation environments and real robot experiments.

Train→Test	Method	PrT	Action Type	VLM	No. Instructions in a Row (1000 chains)					Avg. Len.
					1	2	3	4	5	
ABC→D	Diff-P-CNN [27]	×	Diffusion	×	63.5%	35.3%	19.4%	10.7%	6.4%	1.35±0.05
	Seq-BESO [162]	×	Diffusion	×	76.9%	53.3%	35.8%	24.2%	16.1%	2.06±0.13
	MDT [163]	×	Diffusion	×	63.1%	42.9%	24.7%	15.1%	9.1%	1.55
	RoboFlamingo [105]	×	Cont.	✓	82.4%	61.9%	46.6%	33.1%	23.5%	2.47
	SuSIE [12]	✓	Diffusion	×	87.0%	69.0%	49.0%	38.0%	26.0%	2.69
	DeeR-VLA [216]	×	Cont.	✓	84.8%	72.3%	54.9%	44.6%	33.5%	2.90
	GR-1 [208]	✓	Cont.	×	85.4%	71.2%	59.6%	49.7%	40.1%	3.06
	OpenVLA [89]	✓	Discrete	✓	91.3%	77.8%	62.0%	52.1%	43.5%	3.27
	3DDA [84]	×	Diffusion	×	93.8%	80.3%	66.2%	53.3%	41.2%	3.35
	MoDE [160]	✓	Diffusion	×	96.2%	88.9%	81.1%	71.8%	63.5%	4.01±0.04
	RoboDual [20]	✓	Diffusion	✓	94.4%	82.7%	72.1%	62.4%	54.4%	3.66
	VPP [69]	✓	Diffusion	×	95.7%	91.2%	86.3%	81.0%	75.0%	4.29
	Seer [193]	✓	Cont.	×	96.3%	91.6%	86.1%	80.3%	74.0%	4.28
	FLOWER (ours)	×	Flow	✓	–	–	–	–	–	**4.44±0.04**
	FLOWER (ours)	✓	Flow	✓	–	–	–	–	–	**4.53±0.04**
ABCD→D	Diff-P-CNN [27]	×	Diffusion	×	86.3%	72.7%	60.1%	51.2%	41.7%	3.16±0.06
	Seq-BESO [162]	×	Diffusion	×	93.6%	87.3%	81.0%	74.9%	67.7%	4.05±0.04
	RoboFlamingo [105]	×	Cont.	✓	96.4%	89.6%	82.4%	74.0%	66.0%	4.09
	DeeR-VLA [216]	×	Cont.	✓	99.1%	93.3%	82.1%	74.6%	63.8%	4.13
	GR-1 [208]	✓	Cont.	×	94.9%	89.6%	84.4%	78.9%	73.1%	4.21
	MDT [163]	×	Diffusion	×	98.6%	95.8%	91.6%	86.2%	80.1%	4.52±0.02
	MoDE [160]	✓	Diffusion	×	97.1%	92.5%	87.9%	83.5%	77.9%	4.39±0.04
	FLOWER (ours)	×	Flow	✓	–	–	–	–	–	**4.62±0.03**
	FLOWER (ours)	✓	Flow	✓	–	–	–	–	–	**4.67±0.04**
D→D	Seq-BESO [162]	×	Diffusion	×	89.4%	74.0%	61.1%	51.7%	42.6%	3.19±0.06
	MDT [163]	×	Diffusion	×	93.7%	84.5%	74.1%	**64.4%**	55.6%	3.72±(0.05)
	RoboUniView [111]	✓	Cont	✓	96.2%	88.8%	77.6%	66.6%	56.3%	3.85
	FLOWER (ours)	✓	Flow	✓	–	–	–	–	–	**4.35±0.02**

Table D.5.: CALVIN Benchmark results for D, ABC and ABCD. Columns 1–5 report the percentage of 1000 evaluation chains completing at least the first one through five instructions consecutively. Avg. Len. is the mean number of consecutively completed instructions (maximum 5). Results are reproduced from the published comparison [161]. FLOWER entries retain the published three-seed means and standard deviations; per-instruction rates are not reported here. Standard deviations are shown where reported.

D.3.1. CALVIN Benchmark

For all experiments settings, we train FLOWER for up to 40k steps across 4 GPUS with a batch size of 8 each. The standardized evaluation protocol enable us to directly compare the results of FLOWER against other baselines, which enables a fair comparison to prove the SoTA performance of FLOWER.

Baselines We compare FLOWER against a diverse set of available baselines on all CALVIN variants. Relevant VLAs include RoboDual [20], OpenVLA [89], RoboFlamingo [105] and depth-reconstruction pretrained RoboUniView [111]. In addition we consider video-based policies pretrained on diverse

video data Seer [193], Video-Prediction-Policy [69] and GR-1 [208]. Moreover, we consider relevant diffusion-based policies like MoDE [160], MDT [163] and depth-based 3D Diffuser Actor [84] and SuSIE [12]. FLOWER surpasses all of these baselines across all CALVIN variants with remarkable efficiency in just 6 hours of finetuning.

D.3.2. SIMPLER Benchmark

Method	Put Carrot on Plate	Spoon on Towel	Stack the Blocks	Eggplant in Yellow Basket	Average
RT-1-X	4	0	0	0	1.1
Octo	8	12	0	43	16
CrossFormer	15	15	0	**92**	30
OpenVLA	0	0	0	4	1.0
FLOWER	13	**71**	**8**	88	**45**

Table D.6.: Experimental Results for the SIMPLER Bridge Benchmark. Average Performance comparison across all Tasks in the Bridge Setting.

Method	Open/Close Drawer	Move Near	Open Top Drawer and Place Apple	Pick Coke Can	Average
RT-1-X	**59.7**	31.7	**21.3**	**56.7**	**42.4**
Octo	22.7	4.2	0.0	17.0	11.0
CrossFormer	0.5	4.6	0.0	0.0	1.3
OpenVLA	35.6	**46.2**	0.0	16.3	24.5
FLOWER	27.8	43.3	0.0	**56.3**	31.9

Table D.7.: Experimental Results for the SIMPLER Google Robot Benchmark. Reported full-task success for the four-category FLOWER comparison. The shared baseline entries coincide with the Visual Matching values in Chapter 5; they are not equal-weight averages of Visual Matching and Variant Aggregation. Average summarizes the four displayed categories, including apple placement.

We evaluate FLOWER on the SIMPLER benchmark [106] after pretraining on our cross-embodiment mix. SIMPLER is a real2sim benchmark that implements several scenes from the diverse BridgeV2 [200] and Google Robot setup [18] to test foundation policies after pretraining. The benchmark requires policies to run approximately 3000 rollouts in different settings across two benchmarks with 8 different tasks in various conditions.

Baselines. On this setup, we compare FLOWER against RT-1X [30], Octo [140], OpenVLA [89], and CrossFormer [36]. For each model, we test on the full benchmark and report the average results for a fair comparison.

Results. The results for the Google Robot tasks are summarized in Table D.7 and the results for the Bridge challenge are shown in Table D.6. Overall, FLOWER outperforms both Octo and OpenVLA on both benchmarks, despite having only 200 GPU hours of pre-training on heterogeneous robot datasets. Notably, the FLOWER variant achieves stronger overall performance, with gains on the Bridge Benchmark. In contrast, on the Google Robot benchmark, RT-1X attains the highest performance across several tasks, suggesting that further improvements in action space modeling or pretraining diversity might be beneficial. However, FLOWER achieves the second best performance and surpasses all other generalist policies in this setting.

These findings show that FLOWER delivers strong performance with low computational demands **(RQ II)** as well as robustly handles diverse robot embodiments and action spaces **(RQ III)**. The robust performance on the Bridge Benchmark highlights its capability to manage diverse action spaces after heterogeneous pretraining, whereas the areas of lower performance on the Google Robot benchmark point to opportunities for further refinement.

D.3.3. LIBERO Benchmark

The LIBERO benchmark [110] comprises multiple task suites testing different aspects of robotic manipulation. LIBERO-10 provides 50 demonstrations for 10 tasks, while LIBERO-90 extends to 90 different tasks. Both versions use a Franka Emika Panda robot with end-effector control and dual camera inputs (static and wrist). The benchmark includes five distinct suites: Spatial (testing spatial relationships), Goal (varying objectives), Object (object manipulation), Long (extended task duration), and Suite-90 (diverse short-horizon tasks). Each task is evaluated over 50 trials for each with different starting positions. We finetune FLOWER with 30k training steps on each setting and 50k steps for LIBERO 90 given its increased size.

We compare FLOWER against both generalist policies such as π_0 [11] and π_0-FAST [151], the improved $\pi_{0.5}$-ki (knowledge insulation) [38] OpenVLA [89], OpenVLA-OFT [88] Chain-of-Affordance-VLA (CoA-VLA) [102], Octo [140], MiniVLA as well as against the current state-of-the-art specialist policy Baku [59], which uses a small transformer-based model with action chunking. As shown in Table D.8, FLOWER achieves a competitive four-suite average of 96.9%, compared with OpenVLA-OFT's 97.1%, and leads the Object and Long suites. Its success rates are above 93% across all five suites. It also

remains competitive with $\pi_{0.5}$ a closed source VLA pretrained on a much larger dataset with 3b+ parameters.

	Spatial SR (↑)	Object SR (↑)	Goal SR (↑)	Long SR (↑)	90 SR (↑)	Average (without 90) SR (↑)
Diff-P-CNN [27]	78.3 ± 1.1%	92.5 ± 0.7%	68.3 ± 1.2%	50.5 ± 1.3%	-	72.4 ± 0.7%
Octo [140]	78.9 ± 1.0%	85.7 ± 0.9%	84.6 ± 0.9%	51.1 ± 1.3%	-	75.1 ± 0.6%
OpenVLA [89]	84.7 ± 0.9%	88.4 ± 0.8%	79.2 ± 1.0%	53.7 ± 1.3%	-	76.5 ± 0.6%
OpenVLA-OFT [88]	97.6%	98.4%	**97.9%**	94.5%	-	**97.1%**
CoA-VLA [102]	85.3 ± 0.9%	93.1 ± 1.0%	85.8 ± 0.9%	55.0 ± 1.2%	-	79.8 ± 0.5%
Baku [59]	-	-	-	86.0%	90.0%	-
MiniVLA	-	-	-	-	86.0%	-
MoDE [160]	-	-	-	94.0%	95.0%	-
π_0 [11]	96.8%	98.8%	95.8%	85.2%	-	94.2%
π_0-FAST [151]	96.4%	96.8%	88.6%	60.2%	-	85.5%
$\pi_{0.5}$-ki (from scratch) [38]	96.6%	97.2%	94.6%	84.8%	92.7%	93.3%
$\pi_{0.5}$-ki (from generalist model) [38]	**98.0%**	97.8%	95.6%	85.8%	**96.0%**	94.3%
FLOWER	97.5 ± 0.8%	**99.1 ± 0.4%**	96.1 ± 0.6%	**94.9 ± 1.2%**	94.7 ± 1.0%	96.9 ± 0.7%

Table D.8.: Experimental Results for the LIBERO Benchmarks. SR: Success Rate. Best results in each column are shown in bold. FLOWER leads the Object and Long suites and remains competitive across the other settings.

D.3.4. ALOHA Benchmark

The Aloha benchmark [222] provides 50 human-collected demonstrations for two tasks: Cube Transfer and Insertion. In both tasks, a bi-manual Aloha robot operating in joint space is equipped with a single top-view camera and proprioceptive state input. We evaluate each task over 500 episodes, with each episode consisting of 500 steps. For the baselines we adopt ACT[222] and Diffusion Policy[27] implemented by lerobot [21].

Next, we evaluate FLOWER's ability to learn challenging high frequency control on the Aloha simulation setup [222]. We test several versions of FLOWER that have been finetuned on different pretraining mixes: Cross-X and joint state DROID only. In addition, we compare FLOWER against two common specialist policies: Diffusion Policies and the state-of-the-art policy for bi-manual setups Action Chunking Transformer (ACT) [222]. We use the two simulation tasks, "Insert the peg into the socket." and "Pick up the cube with the right arm and transfer it to the left arm.", that are visualized in Figure 6.4. The dataset contains 50 human-collected demonstrations for each tasks.

Results. As shown in Figure D.1, FLOWER achieves a strong performance on both tasks with eight denoising steps and outperforms the specialist ACT

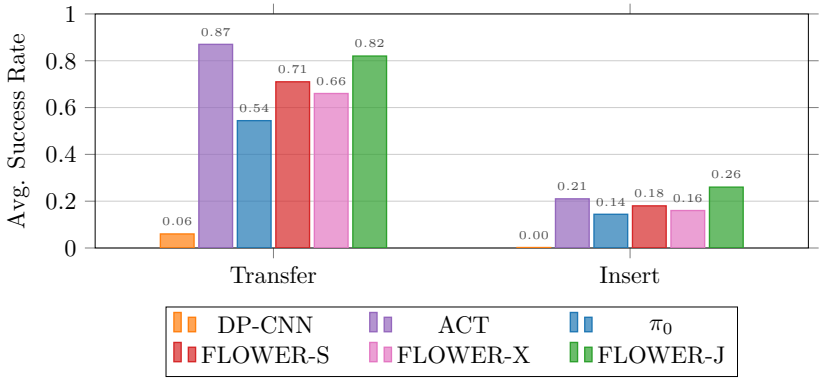


Figure D.1.: ***Aloha Simulation Tasks.*** Our average success rates over 500 evaluations on the *Cube Transfer* and *Bimanual Insertion* simulation tasks introduced by Zhao et al. [222]. *S* denotes FLOWER trained from scratch, *X* applies cross-action space pretraining, and $\mathcal{J}$ represents DROID-only joint state pretraining.

policy on the challenging Insertion task by a considerable margin. FLOWER achieves comparable performance on the Transfer task compared to ACT except for the variant pretrained on cross action space data. The standard Diffusion Policy is not able to solve any of the tasks. Comparing the different pretraining versions of FLOWER, we find that the joint only mix using DROID achieves the best results by a considerable margin. Surprisingly, the cross-embodied pretraining is not able to achieve strong results and its final performance is even lower than the version trained from scratch.

D.3.5. Real Kitchen Play Dataset

We conducted data collection through teleoperation, utilizing a leader-follower robot configuration to ensure precision and intuitive control [80]. The dataset includes proprioceptive sensor readings and images captured by two static cameras at a frequency of 6 Hz. Actions were represented as normalized desired joint positions. The shared BESO/FLOWER collection contains 417 labeled short-horizon segments from approximately 45 minutes of play data, each paired with text instructions. BESO evaluates ten tasks from the kitchen repertoire, while FLOWER evaluates twenty tasks. To

enhance diversity in task descriptions, GPT-4 was employed to generate varied language annotations.

Evaluation Protocol. Each policy is tested 5 times for each task from a starting position not seen in training with some added noise to it. During our experiments, we further varied the orientation of the banana slightly for the robot to pick up, while we kept the toaster in the same position during all our experiments. We report the average success rate and rank for each task to determine the best policy.

D.3.5.1. Pretraining Details for Baselines for the Real World Kitchen

For finetuning the baseline generalist robot policies (Octo, CrossFormer, and OpenVLA), we adhered as closely as possible to the official recommendations provided in their respective GitHub repositories, with only minimal modifications where necessary. All experiments were conducted on a single-node GPU cluster equipped with four NVIDIA RTX 4090 GPUs (24GB each).

For **Octo**, we fine-tuned the model for 50,000 steps, which took approximately 6 hours. In our setup, we used two images per sample—designating the top image as `image_primary` and the side image as `image_secondary`. The baseline Octo model has a default action space of 7 dimensions (delta end-effector position, rotation, and gripper controls). To accommodate our tasks, we extended the default action head by one additional dimension, creating a 7+1 dimensional absolute action space before fine-tuning.

For **CrossFormer**, we again relied on the default fine-tuning settings from the original repository and trained for 50,000 steps, which took around 12 hours. The image setup was identical to that used for Octo. We introduced a new action head, `new_arm_single_joint`, with an action dimension of 8, and developed a new observation tokenizer specifically for the secondary image. All other modules were initialized from the pretrained weights provided in the repository and then fine-tuned.

For **OpenVLA**, which originally supports only delta end-effector actions and enforces an assertion to prevent the use of joint-space OXE datasets for pretraining or fine-tuning, we modified the code to remove this assertion. We then introduced appropriate action and normalization masks (masking nothing for the action and masking the gripper for normalization). Using the default fine-tuning configuration with LoRA-based updates, we trained

OpenVLA for 150,000 steps. Due to memory constraints, we reduced the batch size from the default of 16 to 1 and applied gradient accumulation over 4 steps (as opposed to a larger accumulation factor, which would have substantially increased training time). This fine-tuning process took approximately 60 hours on our 4-GPU setup.

For FLOWER we finetuned our model on the kitchen dataset for 50,000 steps. Since FLOWER has been pretrained on single image, we extended the second static image for finetuning. No additional modifications have been made to guarantee a fair comparison against the baselines.

All baseline models use the same kitchen dataset and GPU setup, with model-specific finetuning schedules: 50,000 steps for Octo and CrossFormer and 150,000 steps for OpenVLA. FLOWER is finetuned for 50,000 steps.

D.3.5.2. Failure Cases for different Policies

Octo. The most common failure mode involves Octo fixating on the microwave - repeatedly opening, closing, or attempting to interact with its door even when the task involves other objects or locations. The second most frequent failure involves Octo's poor object manipulation, particularly with the pot and banana, where it either drops items prematurely or fails to lift them high enough to clear obstacles like the sink edge. Finally, there's a consistent pattern of spatial navigation issues where Octo either pushes objects into walls, hovers aimlessly above target locations, or places objects in incorrect intermediate positions (like between the sink and stove).

CrossFormer. The Crossformer policy exhibits several consistent failure patterns across tasks, including freezing in place, hovering without executing actions, and getting stuck on objects (e.g., sink, microwave door, oven). Many failures involve misinterpreting tasks, such as repeatedly pretending to place toast in the sink or confusing objects like the banana and the oven tray. The model also struggles with manipulating objects correctly, often failing to grasp, dropping, or pushing objects off surfaces rather than placing them accurately. Additionally, it frequently interacts with unintended objects, such as opening and closing the microwave

OpenVLA. OpenVLA frequently fails due to object manipulation errors, such as pushing, flipping, or throwing objects off surfaces rather than placing them correctly. A recurring issue is poor grasping ability, especially with pots and

bananas, often failing to lift them or dropping them prematurely. Additionally, the policy exhibits random movement behaviors, such as hovering aimlessly, crashing into the kitchen, or moving without executing the task. It also struggles with partial execution, frequently opening and then immediately closing doors or trays instead of completing the full action.

FLOWER. The most common failure mode of FLOWER is imprecise spatial positioning, particularly evident in tasks like pushing the toaster lever where the agent consistently misses by about 1cm. We hypothesize that this is due to workspace normalization issues at boundary regions. The second major failure pattern involves interaction with pots in the sink, where FLOWER either gets stuck in loops just before completion, fails to properly clear the sink walls, or incorrectly routes objects (like trying to drop pots into the sink during stove-to-stove transfers). Finally, there are issues with excessive force application in some cases, particularly with the toaster where the agent occasionally rips it off rather than interacting with it properly.

D.3.6. Generalization Experiments

Finally, we evaluate FLOWER against the best baseline in several generalization experiments. In these experiments, we test the models under conditions that introduce variations not encountered during training. Table D.10 reports the performance of the different methods on tasks such as "Move Pot from Right Stove to Sink", "Open Oven", and "Pull Oven Tray" under three scenarios: Novel Object, Flashlight, and Background Distractors. For instance, in the Novel Object condition, new object instances are introduced, while the Flashlight and Background Distractors settings simulate changes in illumination and environmental clutter. These settings collectively challenge the models to generalize beyond their training distribution.

In particular, our experiments also examine the models' abilities to manipulate novel objects—those not present in the initial training distribution. The objects that are new to the model are highlighted in **bold** in Table D.10. As shown, FLOWER consistently achieves higher success rates and lower ranks when manipulating these unfamiliar items, demonstrating robust performance even under significant distribution shifts. Figure 6.7 provides visual examples of these novel objects and the scene with various background distractions. This additional analysis underscores the strength of our approach in adapting to unseen variations in both object appearance and environmental context.

D.3.6.1. Novel Task Compositions

To further evaluate our method's capacity for compositional generalization, we designed a set of novel task compositions that require the agent to combine multiple subtasks into a coherent, long-horizon plan. Each task is defined as a sequence of actions that must be executed in a specific order. For instance, the **Sequence: Open and Close All Appliances** task comprises the following subtasks: "Open the Microwave", "Open the Oven", "Open the Ice", "Close the Ice", "Close the Oven", and "Close the Microwave". This sequence challenges the model to manipulate various kitchen appliances in a coordinated manner, ensuring that the prescribed order of operations is maintained under varying conditions.

In addition, we introduced two other sequence tasks to test different aspects of compositionality. The **Sequence: Move Items Between Stovetop and Sink** task requires the agent to transfer items between workstations, with subtasks including "Move Banana from Right Stove to Sink", "Push the Toaster Lever", "Move Pot from Left Stove to Right Stove", "Pick Up Toast and Place it at the Sink", "Move Pot from Right Stove to Left Stove", and "Move Banana from Sink to Right Stove". Finally, the **Sequence: Operate the Oven** task focuses on oven manipulation and is composed of the subtasks "Open the Oven", "Pull the Oven Tray", "Move Banana from Right Stove to Oven Tray", "Push the Oven Tray", and "Close the Oven". These novel task compositions simulate realistic, multi-step scenarios and provide a rigorous benchmark for evaluating the ability of our approach to integrate learned sub-skills into coherent, long-horizon behaviors.

Task	SR/R	Octo	OpenVLA	CrossFormer	FLOWER
Pot from right stove to sink	SR	0.0%	60.0%	**100.0%**	80.0%
	R	4	3	**1**	2
Pot from sink to right stove	SR	0.0%	0.0%	0.0%	**20.0%**
	R	2	2	2	**1**
Open oven	SR	40.0%	0.0%	0.0%	**60.0%**
	R	2	3	3	**1**
Pull oven tray	SR	0.0%	40.0%	0.0%	**100.0%**
	R	3	2	3	**1**
Open microwave	SR	40.0%	**100.0%**	40.0%	**100.0%**
	R	3	**1**	3	**1**
Close microwave	SR	20.0%	80.0%	40.0%	**100.0%**
	R	4	2	3	**1**
Banana from right stove to sink	SR	10.0%	40.0%	40.0%	**100.0%**
	R	4	2	2	**1**
Banana from sink to right stove	SR	0.0%	0.0%	20.0%	**60.0%**
	R	3	3	2	**1**
Push toaster lever	SR	0.0%	**100.0%**	0.0%	0.0%
	R	2	**1**	2	2
Pickup toast and put to sink	SR	0.0%	20.0%	**80.0%**	40.0%
	R	4	3	**1**	2
Open Ice	SR	0.0%	0.0%	0.0%	**100.0%**
	R	2	2	2	**1**
Banana from right stove to oven tray	SR	0.0%	0.0%	0.0%	**40.0%**
	R	2	2	2	**1**
Pot from sink to left stove	SR	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	R	**1**	**1**	**1**	**1**
Pot from left stove to right stove	SR	40.0%	0.0%	20.0%	**80.0%**
	R	2	4	3	**1**
Banana from tray to right stove	SR	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	R	**1**	**1**	**1**	**1**
Close oven	SR	40.0%	**100.0%**	20.0%	80.0%
	R	3	**1**	4	2
Pot from right stove to left stove	SR	30.0%	0.0%	0.0%	**40.0%**
	R	2	3	3	**1**
Push oven tray	SR	0.0%	20.0%	**40.0%**	20.0%
	R	4	2	**1**	2
Pot from left stove to sink	SR	0.0%	80.0%	40.0%	**100.0%**
	R	4	2	3	**1**
Close Ice	SR	0.0%	0.0%	0.0%	**100.0%**
	R	2.0	2	2	**1**
Overall Performance	SR	10%	31%	22%	**61%**
	R	2.70	2.10	2.20	**1.25**

Table D.9.: Detailed Results for all tested Real Robot Tasks in the Kitchen Environment. Each task has two rows: the first (SR) reports success rate (%), and the second (R) reports rank within that task (lower rank = better performance). The best results per task are highlighted in **bold**.

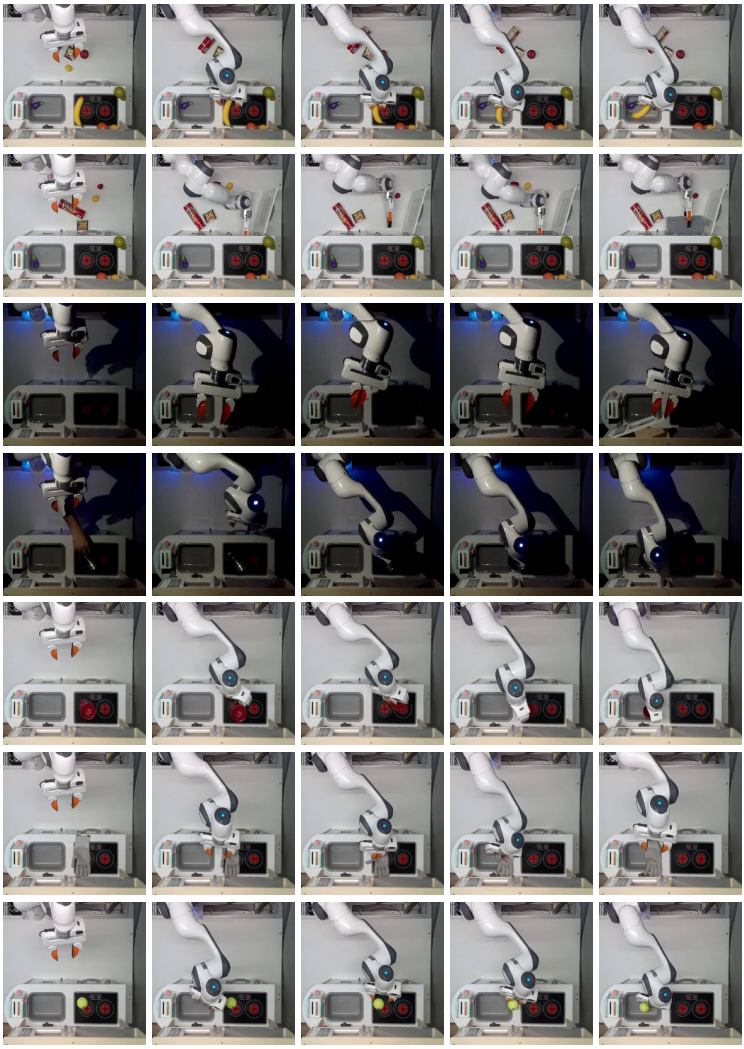


Figure D.2.: Example generalization rollouts. First two rows show rollouts with background distractors, rows 3 and 4 show rollouts with only a flashlight as a light source, and the last 3 rows showcase novel objects.

Task	Novel Object		Flashlight		BG Distractors	
	FLOWER	OpenVLA	FLOWER	OpenVLA	FLOWER	OpenVLA
Move the **black donut** from sink to right stove	**33.3**	0.0	-	-	-	-
Move the **tennis ball** from right stove to sink	**66.7**	33.3	-	-	-	-
Move the **tennis ball** from sink to right stove	0.0	0.0	-	-	-	-
Move the **black donut** from right stove to sink	**33.3**	0.0	-	-	-	-
Move the **red cup** from right stove to sink	33.3	**66.7**	-	-	-	-
Move the **glove** from sink to right stove	**33.3**	0.0	-	-	-	-
Move the **carrot** from right stove to sink	**33.3**	0.0	-	-	-	-
Move the **glove** from right stove to sink	**100**	0.0	-	-	-	-
Move the **carrot** from sink to right stove	0.0	0.0	-	-	-	-
Move the **red cup** from sink to right stove	0.0	0.0	-	-	-	-
Open the microwave	-	-	**100**	**100**	**100**	**100**
Pull the oven tray	-	-	**100**	0.0	**100**	66.7
Move banana from right stove to sink	-	-	**100**	33.3	66.7	66.7
Close the oven	-	-	33.3	**66.7**	66.7	0.0
Push down the toaster lever	-	-	0.0	0.0	33.3	**100**
Move pot from right stove to sink	-	-	-	0.0	**66.7**	33.3
Open the ice box	-	-	0.0	0.0	66.7	0.0
Open the oven	-	-	**100**	0.0	**100**	0.0
Close the microwave	-	-	**100**	**100**	**100**	66.7
Move pot from left stove to sink	-	-	0.0	0.0	33.3	66.7
Close the ice box	-	-	0.0	0.0	**100**	0.0
Pick up toast and put it in the sink	-	-	0.0	0.0	0.0	0.0
Average	**33.3**	10.0	**50.0**	25.0	**69.5**	41.7

Table D.10.: Generalization experimental results for novel objects, distractions and new lighting conditions. The table reports the success rate (in %) of the corresponding policy evaluated under three different generalization scenarios: Novel Object, Flashlight, and Background Distractors (evaluated 3 times for each setting). The best score for each test is highlighted in **bold**. A dash (-) indicates that the task was not evaluated in that scenario.

Task	Method	1	2	3	4	5	Avg. Seq. Len.
Seq: Stovetop + Sink	FLOWER	**66.7%**	**66.7%**	**66.7%**	**33.3%**	**33.3%**	**2.67**
	OpenVLA	**66.7%**	33.3%	33.3%	0.0%	–	1.33
Seq: Open Close All	FLOWER	**100%**	**100%**	**100%**	**100%**	**100%**	**5.00**
	OpenVLA	33.3%	0.0%	–	–	–	0.33
Seq: Oven	FLOWER	0.0%	–	–	–	–	0.00
	OpenVLA	0.0%	–	–	–	–	0.00
Overall Performance (FLOWER)		**51.1%**	–	–	–	–	**2.56**
Overall Performance (OpenVLA)		**16.7%**	–	–	–	–	**0.55**

Table D.11.: Long Horizon Task Composition Results. For each sequence task, the per-instruction success rates (in %) are shown for the first 5 instructions (if applicable) along with the average sequence length. "–" indicates that no instruction was successfully solved at that index. The Overall Performance rows report the average success rate (computed over all available instructions) and the average sequence length across tasks for each method.