Interpretable Representation Learning for Motion Forecasting

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

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

genehmigte

Dissertation

von

Royden Wagner

Tag der mündlichen Prüfung:	11.12.2025
1. Referent:	Prof. Dr.-Ing. Christoph Stiller
2. Referent:	Prof. Dr. Abhinav Valada

Usage of large language models

To enhance the written presentation of this thesis, I used large language models (LLMs), including Claude Sonnet 4, perplexity pro, ChatGPT 4o, and DeepL. I used these models to linguistically and grammatically enhance and restructure individual sentences and text passages. All LLM-generated text was manually checked and often revised further. The models were not used to generate new content. All methods, experiments, and results described in this thesis were developed independently or by the co-authors indicated in each case (see Preface).

Abstract

We address the challenge of interpretable representation learning for motion forecasting in self-driving vehicles. Instead of treating transformer models as opaque black boxes, we develop methods to interpret and modify learned representations. Our work builds upon ideas from self-supervised learning, mechanistic interpretability methods, and mixture density networks.

We introduce two interpretable pre-training methods, which leverage redundancy reduction and multimodal similarity learning. Specifically, we improve prediction accuracy and accelerate training by maximizing the similarity of embeddings from augmented environments and aligning motion with scene context.

Moving beyond pre-training, we probe the latent space of motion forecasting models and reveal human-interpretable features such as speed or agent type.

We find the learned vector representation of these features, enabling us to modify model outputs through targeted interventions. This allows us to compensate for domain shifts like slower driving styles and to interpret model behavior.

We further find retrocausal mechanisms in forecasting models, which connect later parts of trajectories to earlier parts. We leverage these mechanisms to issue goal-based and directional instructions. Remarkably, regular training for motion forecasting without instructions leads to the ability to follow and adapt such instructions to the scene context.

Our empirical evaluations on state-of-the-art benchmarks, including Waymo Open Motion and Argoverse 2 Forecasting, confirm that our methods advance both accuracy and interpretability, bridging the gap between high-performing models and those whose underlying mechanisms can be interpreted and influenced.

Kurzfassung

Diese Arbeit befasst sich mit Methoden des maschinellen Lernens zur Bewegungsvorhersage von Verkehrsteilnehmern. Der Fokus liegt dabei auf der Analyse und Interpretation der Mechanismen, die Transformer-Modelle lernen.

Zunächst werden zwei interpretierbare Trainingsmethoden beschrieben, die gelernte Vektor-Repräsentationen von verschiedenen Modalitäten aneinander angleichen. Diese verbessern die Vorhersagegenauigkeit und beschleunigen das Training, indem sie die Repräsentationen für ähnliche Straßenszenen angleichen und Bewegungen mit dem Szenenkontext abstimmen.

Darüber hinaus werden die gelernten Vektor-Repräsentationen von Vorhersagemodellen auf interpretierbare Merkmale (d.h., Features) untersucht. Diese Arbeit zeigt dabei die gelernten Repräsentationen für Bewegungs-Features wie Geschwindigkeit oder Richtung. Dadurch können gelernte Mechanismen interpretiert und Vorhersagen an z.B. langsamere Fahrstile in einer neuen Umgebung angepasst werden.

Außerdem deckt diese Arbeit retrokausale Mechanismen in Vorhersagemodellen auf, die spätere Teile von Trajektorien mit früheren Teilen verbinden. Diese Mechanismen können genutzt werden, um zielbasierte und richtungsweisende Instruktionen zu geben. Bemerkenswert ist, dass das Training ohne Instruktionen dazu führt, dass solche Instruktionen dennoch befolgt und an den Szenenkontext angepasst werden.

Die empirische Evaluation anhand moderner Datensätze, darunter Waymo Open Motion und Argoverse 2 Forecasting, bestätigt, dass die vorgestellten Methoden sowohl die Genauigkeit als auch die Interpretierbarkeit verbessern. Damit trägt diese Arbeit zur Entwicklung von performanten Vorhersagemodellen bei, deren zugrunde liegende Mechanismen interpretiert und beeinflusst werden können.

Preface

First and foremost, I would like to express my gratitude to my main advisor, Prof. Christoph Stiller. I am grateful for the trust you placed in me, the opportunity to join your research institute, and the guidance that led to several research papers. I also deeply appreciate the generous resources, which you provided for experiments, research vehicles, conference travels, and other necessities.

I extend my appreciation to the research group leaders, Dr.-Ing. Omer Sahin Tas, Dr. Carlos Fernandez, and Dr. Martin Lauer. Many thanks to Sahin for sparking my interest in mechanistic interpretability in our weekly meetings on representation learning and countless hours of collaborating on our joint papers (see below). Many thanks to Carlos and Martin for continuous and valuable feedback in the meetings of the perception and prediction group.

Many thanks to Prof. Abhinav Valada for joining my doctoral jury. Moreover, I appreciate the valuable feedback on my thesis topic by Prof. Anne Koziolek, Jun.-Prof. Maike Schwammberger, and Prof. Hannes Hartenstein.

Furthermore, I am grateful to my other collaborators, in alphabetical order: Felix Hauser, Christian Kinzig, Marvin Klemp, Fabian Konstantinidis, Dr.-Ing. Hendrik Königshof, Kevin Rösch, Yinzhe Shen, Marlon Steiner, Dominik Strutz, Dr.-Ing. Jannik Quehl, Jaime Villa, Abhishek Vivekanandan, Kaiwen Wang, and Dr.-Ing. Florian Wirth. Special thanks to Marvin for co-authoring 7 papers.

Last but not least, many thanks to my family and friends for supporting me on this doctoral journey, which started out as a bit of an Odyssey, with early research at SAP and later at Heidelberg University. Especially many thanks to my parents, Ulyka and Günter, for having patience with me throughout this time.

While the research presented here bears my name, it builds upon the excellent collaboration with my co-authors and advisors. Specifically, this doctoral thesis is based on four recent papers. In the following list, first-authors and joint first-authors are highlighted in bold and individual contributions are listed below:

1. **Royden Wagner**, Omer Sahin Tas, Marvin Klemp, Carlos Fernandez, and Christoph Stiller. RedMotion: Motion Prediction via Redundancy Reduction. In *Transactions on Machine Learning Research (TMLR)*, 2024

   Project idea: R.W.; codebase: R.W.; experiments: R.W.; paper writing: R.W., O.S.T., and M.K.; guidance, funding, and infrastructure: O.S.T., C.F., and C.S.

2. **Royden Wagner**, **Omer Sahin Tas**, Marvin Klemp, and Carlos Fernandez. JointMotion: Joint Self-Supervision for Joint Motion Prediction. In *Conference on Robot Learning (CoRL)*, 2024

   Project idea: R.W. and O.S.T.; codebase: R.W.; experiments: R.W.; paper writing: R.W., O.S.T., and M.K.; guidance, funding, and infrastructure: O.S.T. and C.F.

3. **Omer Sahin Tas** and **Royden Wagner**. Words in Motion: Extracting Interpretable Control Vectors for Motion Transformers. In *International Conference on Learning Representations (ICLR)*, 2025

   Project idea: O.S.T and R.W.; codebase: R.W. and O.S.T.; experiments: R.W. and O.S.T.; paper writing: O.S.T and R.W.; guidance, funding, and infrastructure: O.S.T.

4. **Royden Wagner**, Omer Sahin Tas, Felix Hauser, Marlon Steiner, Dominik Strutz, Abhishek Vivekanandan, Carlos Fernandez, and Christoph Stiller. RetroMotion: Retrocausal Motion Forecasting Models are Instructable. In *arXiv*, 2025

   Project idea: R.W.; codebase: R.W., F.H., and M.S.; experiments: R.W., F.H., M.S., A.V., and D.S.; paper writing: R.W., O.S.T., F.H., D.S., and A.V.; guidance, funding, and infrastructure: O.S.T., C.F., and C.S.

Most of the content is edited to fit a consistent storyline, yet some content is reused without changes. Furthermore, this thesis extends the content of the aforementioned papers in the following ways:

- This thesis connects pre-training with interpretable objectives and methods for mechanistic interpretability. Specifically, such pre-training is a step toward mechanistic interpretability as it provides a training setting in which interpretable representations and mechanisms are more likely to be learned, see Chapter 3 and Chapter 4.
- This work provides a detailed mathematical description of the PCA-based pooling for control vectors, see Section 4.2. It further connects neural collapse and controllability with control vectors, see Section 4.5. Moreover, Section 4.6 covers fuzz testing of control vector temperatures for our speed control vectors.
- This thesis extends the description of exponential power distributions, including probability density functions and plots that illustrate the effect of the shape parameter, see Section 5.1.1. Furthermore, it includes ablation studies on the sensitivity of RetroMotion w.r.t. the number of coefficients in discrete cosine transforms (see Section 5.2.4) and shows that the model generalizes from 2 to 8 agents (see Section 5.2.5).

Contents

Acronyms and notation

Acronyms

AV2F	Argoverse 2 Forecasting
CDNV	Class-distance normalized variance
CLIP	Contrastive language-image pre-training
CME	Connecting motion and environments
ConvSAE	Convolutional sparse autoencoder
DCT	Discrete cosine transform
FFN	Feed-forward network
GNN	Graph neural network
GPT	Generative pre-trained transformer
GPU	Graphics processing unit
mAP	Mean average precision
minADE	Minimum average displacement error
minFDE	Minimum final displacement error
IDCT	Inverse discrete cosine transform
JumpReLU	Jumping rectified linear units

KoopmanAE	Koopman autoencoder
LLM	Large language model
MCL	Map contrastive learning
MLP	Multi-layer perceptron
MPM	Masked polyline modeling
MR	Miss rate
NLL	Negative log-likelihood
NMS	Non-maximum suppression
NRC	Neural regression collapse
PCA	Principal component analysis
R^2	Coefficient of determination
ReLU	Rectified linear units
SAE	Sparse autoencoder
S-idx	Straightness index
SMoE	Sparse mixture of experts
TMCL	Trajectory-map contrastive learning
OR	Overlap rate
ORP	On-road probability
WTA	Winner-takes-all

Notation

We use the notation of Goodfellow et al. (2016) for vectors, matrices, indexing, and functions.

$\boldsymbol{v}$	A vector
v_i	Element i of vector $\boldsymbol{v}$
v_{-1}	Last element of vector $\boldsymbol{v}$
$\boldsymbol{M}$	A matrix
$\boldsymbol{M}^\top$	Transpose of matrix $\boldsymbol{M}$
$\boldsymbol{M}_{i,:}$	Row i of matrix $\boldsymbol{M}$
$\boldsymbol{M}_{:,j}$	Column j of matrix $\boldsymbol{M}$
$M_{i,j}$	Element i, j of matrix $\boldsymbol{M}$
$\boldsymbol{M}^{(b)}$	b-th matrix of a batch of matrices
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parameterized by $\boldsymbol{\theta}$
$\mathcal{D}(\cdot)$	A probability density function

1 Introduction

Motion in traffic scenarios varies in complexity based on the individual behavior of road users and scene context, such as road markings and traffic light states. Motion forecasting predicts the future motion of road users, like vehicles, pedestrians, or cyclists (see Figure 1.1). It serves as a form of world modeling since motion is a fundamental aspect of the environment surrounding a self-driving vehicle (Baniodeh et al. 2025). Therefore, realistic motion forecasting is crucial to plan safe maneuvers according to the forecasted motion of other road users (Taş et al. 2023, Geisslinger et al. 2023, Bouzidi et al. 2024).

Early in a forecast, motion is largely influenced by past motion, due to the laws of physics. As we move further into the future, this causal influence diminishes and more variations of motion become likely. Recent methods for motion forecasting (Ngiam et al. 2022, Shi et al. 2022, Cui et al. 2023) address this with multiple choice learning (Guzman-Rivera et al. 2012, Lee et al. 2016), where choices are trajectories of future positions. In complex urban scenarios, the distribution over future trajectories is typically multimodal, while trajectories on highways tend to be more unimodal (i.e., uniform).

When modeling the joint trajectory distribution over multiple road users (i.e., agents), the output space grows exponentially with the number of agents. Thus, a common simplification is to model future motion with marginal trajectory distributions per agent (Nayakanti et al. 2023, Zhou et al. 2023a, Zhang et al. 2023). To improve modeling interactive behavior, some works (Luo et al. 2023, Seff et al. 2023, Jiang et al. 2023) forecast joint trajectory distributions for pairs of agents.

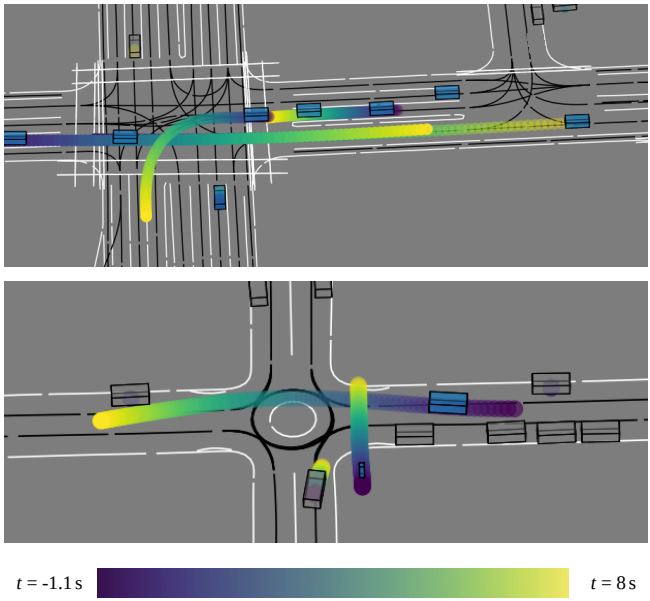


Figure 1.1: Motion forecasting. The forecasts are based on 1.1 s of past motion and cover the next 8 s. Dynamic road users are blue, lanes are black lines, and road markings are white lines.

In this work, we present several methods for interpretable representation learning for motion forecasting. We evaluate our methods on recent motion forecasting datasets, like Waymo Open Motion (Ettinger et al. 2021) and Argoverse 2 Forecasting (Wilson et al. 2023). However, our work differs from purely benchmark- or application-driven approaches to motion forecasting (e.g., Varadarajan et al. 2022, Nayakanti et al. 2023, Zhou et al. 2023a). Such approaches include architectural ablations of new models, but focus on their impact on benchmark performance, not on learned mechanisms.

Research on mechanistic interpretability, on the other hand, analyzes and interprets *how* learned models work. Particularly, learned mechanisms and representations are studied. Mechanistic interpretability methods do not enforce interpretability (e.g., by handcrafting certain rules or mechanisms), but instead

attempt to reverse-engineer learned mechanisms (Chughtai et al. 2023, Nanda et al. 2023, Kitouni et al. 2024). This is broadly related to functionalism in the philosophy of mind (Levin 2023), which models the mind as functions rather than structures, including memory, perceptual, and executive functions[1].

Our work reveals the vector representation of interpretable features (e.g., speed, direction, or agent type) learned by motion forecasting models and shows their impact on forecasts. Specifically, we modify learned representations at inference and analyze the resulting changes in motion forecasts. This enables mechanistic interpretations and generalization to unseen dataset characteristics, like slower driving styles.

Moreover, we find retrocausal mechanisms in forecasting models, which are directed from later parts of trajectories to earlier parts. We leverage these mechanisms to issue goal-based and directional instructions. Notably, regular training for motion forecasting (without instructions) leads to the ability to follow goal-based instructions and to adapt directional instructions to the scene context.

Mechanistic interpretability research can lead to improved benchmark performance as well, yet shifts the perspective from global effects to more fine-grained mechanisms. For example, Zhu et al. (2025) find that normalization layers in transformer models learn tanh-like mappings and improve benchmark performance by replacing normalization with simpler tanh layers. We show that representing a trajectory as sum of cosine functions is well adapted to transformer models, which use cosine-like positional encodings, and improves forecasting precision.

Furthermore, we view self-supervised pre-training with interpretable objectives as a step from enforcing interpretability toward mechanistic interpretability. Such pre-training objectives are based on interpretable proxy tasks yet particularly broad ones with few application-specific details. For example, masked autoencoding is used as pre-training in language modeling (Devlin et al. 2019), computer vision (He et al. 2022), and motion forecasting (Cheng et al. 2023). Hence, the goal of

[1] Similarly, in machine learning, structures are the type, number, and size of layers, while their functions are the learned mechanisms represented by the corresponding weights and biases.

self-supervised pre-training is to learn representations and mechanisms that are a good initialization (i.e., staring point) for a wide range of applications.

We present two pre-training objectives for self-supervised learning in the context of motion forecasting. The first is to learn representations of the environment in traffic scenarios, which are invariant to moderate augmentations. This is motivated by the fact that motion in similar environments tends to be similar as well. Our second objective extends masked autoencoding by learning consistent global representations from inputs of different modalities. Specifically, the similarity of embeddings of motion and map data is maximized as pre-training. This is inspired by the finding that cross-modal pre-training leads to good representations for downstream applications, as shown by CLIP (Radford et al. 2021).

In summary, the main contributions of this work are:

- We propose two interpretable pre-training objectives for self-supervised learning in the context of motion forecasting. Both objectives combined with supervised fine-tuning improve forecasting precision and generalize across different model architectures.
- We reveal the vector representation of interpretable features learned by motion forecasting models. This enables (1) modifying learned representations at inference and (2) to formulate mechanistic interpretations of the influence of these features. Furthermore, we use such modifications for generalization to unseen dataset characteristics, such as different driving styles.
- We find retrocausal mechanisms in forecasting models, which connect later parts of trajectories to earlier parts. We leverage these mechanisms to issue goal-based and directional instructions. Remarkably, regular training for motion forecasting without instructions leads to the ability to follow and adapt such instructions to the scene context.

2 Fundamentals and related work

This chapter includes fundamentals and related work on motion forecasting and representation learning. Specifically, Section 2.1 covers motion forecasting for self-driving vehicles, while Section 2.2 includes details on self-supervised learning, mechanistic interpretability, sparse dictionary learning, neural collapse, ensemble learning, and transformer models.

2.1 Motion forecasting for self-driving vehicles

This section covers related work on motion forecasting for self-driving vehicles. We start by describing different ways of modeling trajectory distributions, including marginal, conditional, and joint distributions. Afterwards, we discuss details on metrics, datasets, context representations, and reference frames used for motion forecasting. Finally, we describe probabilistic and compressed trajectory representations.

2.1.1 Marginal motion forecasting

A common simplification is to model future motion with marginal trajectory distributions per agent (Nayakanti et al. 2023, Cui et al. 2023, Zhang et al. 2023). These methods process the past motion of surrounding agents as context, but model each agent individually in the output space. The majority of recent marginal motion forecasting methods are transformer-based (e.g., Nayakanti et al. 2023, Zhou et al. 2023a, Zhang et al. 2023), with fewer methods using GNNs (e.g., Cui

et al. 2023). Following the evaluation protocol of recent forecasting benchmarks (Ettinger et al. 2021, Wilson et al. 2023), 6 trajectories per agent are typically forecasted and non-maximum suppression (NMS) is used to generate 6 trajectories with larger ensembles (e.g., Varadarajan et al. 2022, Nayakanti et al. 2023, Zhou et al. 2023a).

2.1.2 Conditional motion forecasting

Related methods (Mangalam et al. 2020, Gilles et al. 2022) extend marginal forecasting models with auxiliary goal prediction[1] and condition trajectory forecasts on predicted goals of surrounding agents. Another approach to conditional motion forecasting is to perform marginal forecasting for the first agent and iteratively condition subsequent forecasts on each other (Tolstaya et al. 2021, Sun et al. 2022, Wirth 2023).

2.1.3 Joint motion forecasting

When modeling the joint trajectory distribution over all agents in a scenario, the output space grows exponentially with the number of agents. Therefore, related methods (Luo et al. 2023, Seff et al. 2023, Jiang et al. 2023) model joint trajectory distributions for pairs of interacting agents. Jiang et al. (2023) perform joint motion forecasting as denoising diffusion process and denoise sets of noisy trajectories conditioned on the scene context. Seff et al. (2023) cast joint motion forecasting as language modeling and learn a vocabulary of discrete motion vectors using autoregressive roll-outs for two agents. Another line of work (Casas et al. 2020, Girgis et al. 2022, Ngiam et al. 2022, Zhou et al. 2023b) uses global latent variables to reduce the modeling burden of a full joint distribution of all per-agent

[1] Also referred to as endpoint prediction (Mangalam et al. 2020) and related to dense prediction (Shi et al. 2022, 2024), where goals are predicted as initial trajectories.

variables. Typically, such global latent variables are represented as query vectors and processed by transformer modules.

We perform both marginal and joint forecasting with one model, by using marginal forecasts as initialization for joint forecasts, see Chapter 5. This includes a retrocausal flow of information and allows us to efficiently forecast motion in more complex scenarios with more than two modeled agents. Furthermore, this enables us to issue goal-based and directional instructions by modifying marginal trajectory distributions.

2.1.4 Motion forecasting metrics

Recent motion forecasting benchmarks (Ettinger et al. 2021, Houston et al. 2021, Wilson et al. 2023) evaluate trajectory predictions based on displacement errors. Trajectory predictions include trajectories, associated probabilities, and optionally positional uncertainties (see Section 2.1.7).

Conceptually, motion trajectories are temporal sequences of x- and y-coordinates. We represent trajectories as matrices. The matrix rows are the temporal dimension and the columns are the spatial dimension of the x- and y-coordinates

$$\boldsymbol{Y} = \begin{bmatrix} Y_{1,1}, Y_{1,2} \\ Y_{2,1}, Y_{2,2} \\ \vdots \\ Y_{T,1}, Y_{T,2} \end{bmatrix} \in \mathbb{R}^{T \times 2}, \tag{2.1}$$

with the number of time steps T.

The most common motion forecasting metrics are the minimum average displacement error (minADE) and the minimum final displacement error (minFDE).

For predicted marginal trajectory distributions, the minADE computes the L_2-norm between the ground truth trajectory $\hat{\boldsymbol{Y}}$ and the closest of K predicted trajectories $\boldsymbol{Y}^{(k)}$

$$\text{minADE} = \frac{1}{T} \arg\min_k \sum_t \left\| \hat{\boldsymbol{Y}}_{t,:} - \boldsymbol{Y}^{(k)}_{t,:} \right\|_2, \tag{2.2}$$

with temporal index $t \in \{1, \dots, T\}$ and trajectory index $k \in \{1, \dots, K\}$.

The minFDE is closely related, but only computed for one time step T

$$\text{minFDE} = \arg\min_k \left\| \hat{\boldsymbol{Y}}_{T,:} - \boldsymbol{Y}^{(k)}_{T,:} \right\|_2. \tag{2.3}$$

For joint trajectory distributions of multiple agents, these metrics are averaged over the agent dimension as well

$$\text{minADE}_{\text{joint}} = \frac{1}{AT} \arg\min_k \sum_a \sum_t \left\| \hat{\boldsymbol{Y}}^{(a)}_{t,:} - \boldsymbol{Y}^{(k,a)}_{t,:} \right\|_2, \tag{2.4}$$

$$\text{minFDE}_{\text{joint}} = \frac{1}{A} \arg\min_k \sum_a \left\| \hat{\boldsymbol{Y}}^{(a)}_{T,:} - \boldsymbol{Y}^{(k,a)}_{T,:} \right\|_2, \tag{2.5}$$

with agent index $a \in \{1, \dots, A\}$.

Wilson et al. (2023) additionally use the Brier-minFDE, which accounts for the predicted probability p per trajectory in marginal forecasts or per trajectory set over all modeled agents in joint forecasts

$$\text{Brier-minFDE} = \text{minFDE} + (1 - p)^2. \tag{2.6}$$

Ettinger et al. (2021) compute three more metrics, the overlap rate (OR), the miss rate (MR), and the mean average precision (mAP).

The OR metric describes the overlap rate between the predicted trajectories with the highest corresponding probability and with all other agents, which were visible

at the start of the forecast. They compute box intersection of 3D bounding boxes for each future time step and count the number of intersections. The final score is computed as number of intersections N_{inter} divided by the number of predicted trajectory distributions

$$\text{OR} = N_{\text{inter}}/N_{\text{distr}}. \tag{2.7}$$

Hence, the score is expected to be higher for joint forecasts than for marginal forecasts, since joint forecasting predicts fewer distributions that cover more agents.

The MR metric counts misses on the evaluated dataset split. Ettinger et al. (2021) define a miss as a prediction, where no trajectory is within a lateral and a longitudinal threshold. They evaluate this at three distinct time steps using the thresholds in Table 2.1.

	Lateral thresh.	**Longitudinal thresh.**
$t = 3\,\text{s}$	$1\,\text{m}$	$2\,\text{m}$
$t = 5\,\text{s}$	$1.8\,\text{m}$	$3.6\,\text{m}$
$t = 8\,\text{s}$	$3\,\text{m}$	$6\,\text{m}$

Table 2.1: Lateral and longitudinal thresholds of the miss rate (MR) metric

Furthermore, they scale these thresholds according to the initial speed s of an agent with

$$\text{Scale}(s) = \begin{cases} 0.5 & \text{if } s < 1.4\,\text{m/s} \\ 0.5 + 0.5\left(\frac{s-1.4}{11-1.4}\right) & \text{if } 1.4\,\text{m/s} \leq s \leq 11\,\text{m/s} \\ 1 & \text{if } s > 11\,\text{m/s}. \end{cases} \tag{2.8}$$

The final score is the number of misses N_{miss} divided by the number of predicted trajectory distributions

$$\text{MR} = N_{\text{miss}}/N_{\text{distr}}. \tag{2.9}$$

Thus, as for the OR metric, the MR score is expected to be higher for joint forecasts than for marginal forecasts.

The mAP metric for motion forecasting is inspired by the mAP metric for object detection (Lin et al. 2014). Ettinger et al. (2021) first assign each ground truth trajectory to a trajectory bucket[2]. The buckets are based on high-level maneuvers and include straight, straight-left, straight-right, left, right, left u-turn, right u-turn, and stationary. Afterwards, they classify whether a trajectory is a miss using the same logic as for the MR metric. Each trajectory classified as miss is treated as a false positive and the others as true positives. Finally, they generate precision-recall curves per bucket and compute the mean area under the curve over all buckets. The Soft mAP metric is a variation that does not penalize false positives. For further details refer to Ettinger et al. (2021).

We additionally compute jerk and tortuosity to quantify how realistic trajectory forecasts are. The average jerk of a trajectory is computed with

$$\text{average jerk} = \frac{1}{T} \sum_{t} \left\| \frac{\Delta^3 \boldsymbol{Y}_{t,:}}{\Delta t^3} \right\|. \tag{2.10}$$

The tortuosity of a trajectory is the total length of a trajectory divided by the Euclidean distance between the first and last trajectory point,

$$\text{tortuosity} = \frac{\sum_{t=2}^{T} \|\boldsymbol{Y}_{t,:} - \boldsymbol{Y}_{t-1,:}\|}{\|\boldsymbol{Y}_{-1,:} - \boldsymbol{Y}_{1,:}\|}. \tag{2.11}$$

2.1.5 Motion forecasting datasets

We evaluate our methods on two large-scale motion forecasting datasets, the Waymo Open Motion Dataset (Ettinger et al. 2021) (abbrv. *Waymo*) and the Argoverse 2 Motion Forecasting Dataset (Wilson et al. 2023) (abbrv. *AV2F*).

The Waymo dataset contains 103,000 scenarios of urban and suburban driving, each 20 seconds long and sampled at 10Hz, totaling about 574 hours of driving

[2] For joint predictions, the trajectory of the first agent is used to determine a bucket for the others as well.

data. It was recorded in six US cities, San Francisco, Los Angeles, Phoenix, Detroit, Mountain View, and Seattle. The datset features more than 10.8M moving objects (vehicles, pedestrians, and cyclists), providing 3D bounding boxes, tracking IDs, and scenario labels for nuanced interactions like unprotected turns or merges. The selected scenarios are supported by detailed 3D map data including lane and road attributes and were specifically mined for interesting behaviors and interaction-heavy events. The official 486,995 training, 44,097 validation, and 44,920 testing scenarios are sampled in a partially overlapping manner and contain $1.1\,\mathrm{s}$ of past and $8\,\mathrm{s}$ of target (i.e., future) data.

The AV2F dataset contains 250,000 scenarios, each 11 seconds long and sampled at 10Hz, totaling about 763 hours of annotated urban driving data. It was also recorded in six US cities, Miami, Austin, Washington D.C., Pittsburgh, Palo Alto, and Detroit. The scenarios feature detailed HD maps, track histories for object location, heading, velocity, and type. We follow Zhang et al. (2023) and remap six object types to match the three agent types of the Waymo dataset. Specifically, we combine agents of type vehicle and bus to a common vehicle type, cyclists, motorcyclists, and riderless bicycle to the cyclist type, and reuse the pedestrian type. Following Zhang et al. (2023), we add default bounding box sizes per agent type (vehicle $4.7\,\mathrm{m} \times 2.1\,\mathrm{m} \times 1.7\,\mathrm{m}$, bus $11\,\mathrm{m} \times 3\,\mathrm{m} \times 3.5\,\mathrm{m}$, pedestrian $0.85\,\mathrm{m} \times 0.85\,\mathrm{m} \times 1.75\,\mathrm{m}$, and cyclist $2\,\mathrm{m} \times 0.8\,\mathrm{m} \times 1.8\,\mathrm{m}$). Besides that, we use the official 200,000 training, 25,000 validation, and 25,000 testing scenarios, which are non-overlapping and contain $5\,\mathrm{s}$ of past and $6\,\mathrm{s}$ of future data.

2.1.6 Scene context and reference frames

In motion forecasting, scene context is modeled using scene-centric or agent-centric reference frames and the corresponding coordinate systems.

Scene-centric approaches (Ngiam et al. 2022, Girgis et al. 2022) model all agent states (past and future) and scene elements (road markings, lanes, and traffic light states) using a shared global reference frame. The corresponding coordinate

system is typically centered at the current position of the self-driving vehicle[3]. This allows for efficient processing since all inputs are encoded only once per scene. Furthermore, this enables modeling interactions among agents in a shared coordinate system.

Agent-centric approaches (Varadarajan et al. 2022, Shi et al. 2022, Nayakanti et al. 2023, Wang et al. 2023b) transform all scene elements and past states of surrounding agents into local coordinate systems. Each local coordinate system is centered at the current position of the modeled agent. This leads to repeated computation in scenes with many agents close to each other. However, agent-centric modeling is pose invariant (i.e., motion with the same velocity is always represented the same), which eliminates the need to learn geometric transformations from global coordinates. Furthermore, agent-centric modeling is more data-efficient since it generates multiple training samples per scene (one per modeled agent) compared to a single sample per scene in scene-centric modeling. Therefore, agent-centric forecasting models tend to achieve higher accuracy in benchmarks (cf. Su et al. 2022, Wagner et al. 2024b).

In addition, pairwise-relative (Cui et al. 2023, Zhang et al. 2023) and query-centric approaches (Zhou et al. 2023a,b, Shi et al. 2024) model motion using local reference frames and scene elements using global reference frames.

We use local agent-centric views for their improved data efficiency and a latent context module to model interactions in a shared global latent space, see Chapter 5.

2.1.7 Probabilistic trajectory representations

A common method (e.g., Shi et al. 2022, Zhou et al. 2023a, Zhang et al. 2023) for probabilistically modeling positional uncertainty are mixture density networks (Bishop 1994). Mixture density networks are trained using maximum likelihood

[3] Also referred to as ego-vehicle or ego-agent (Girgis et al. 2022).

estimation, which is rooted in frequentist statistics[4]. Specifically, the learning objective is to maximize the likelihood of observing the training data given the model parameters $\boldsymbol{\theta}$. The likelihood formulation for a mixture of assumed probability densities $\mathcal{D}$ is

$$\mathcal{L}_t(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) = \prod_n \sum_k m_k(x_n; \boldsymbol{\theta}) \cdot \mathcal{D}\Big(y_n \mid \boldsymbol{\phi}_{t,k}(x_n; \boldsymbol{\theta})\Big), \tag{2.12}$$

with temporal index $t \in \{1, ..., T\}$, targets $\boldsymbol{y}$, inputs $\boldsymbol{x}$, sample index $n \in \{1, ..., N\}$, mixture weights $\boldsymbol{m}$, mixture index $k \in \{1, ..., K\}$, and density parameters $\boldsymbol{\phi}$. As output constraints, mixture weights are normalized using a $\mathrm{softmax}$ operation and density scale parameters are clamped to positive values. Note that the mixture weights are constant for all temporal indices, while the density parameters change.

To use gradient decent-based optimizers, recent methods minimize the negative log-likelihood during training

$$\mathrm{NLL}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) = -\sum_t \ln\Big(\mathcal{L}_t(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})\Big). \tag{2.13}$$

This loss formulation is prone to mode collapse, where all mixture components converge to similar or even the same trajectory. Therefore, related methods (e.g., Varadarajan et al. 2022, Nayakanti et al. 2023, Zhou et al. 2023b) optimize this objective only for the predicted trajectory closest to the ground truth[5]. In multiple choice learning, this is referred to as winner-takes-all (WTA) loss (Guzman-Rivera et al. 2012, Lee et al. 2016). Alternative solutions are top-k (Makansi et al. 2019) or annealing-based trajectory selection (Xu et al. 2024) during training.

[4] Maximum likelihood estimation is equivalent to Bayesian maximum a posteriori estimation with a uniform prior (i.e., without incorporating prior knowledge).

[5] The closeness to the ground truth is typically measured by the mean L_2-distance (Su et al. 2022).

Typical choices for probability densities are the densities of normal (Varadarajan et al. 2022, Shi et al. 2022, Nayakanti et al. 2023) or Laplace distributions (Zhou et al. 2023a,b).

We propose to model positional uncertainty with variably-shaped exponential power distributions[6], resulting in higher forecasting accuracy than with normal or Laplace distributions, see Chapter 5.

2.1.8 Compressed trajectory representations

Most motion forecasting methods regress trajectories at a frequency of 10 Hz (e.g., Zhou et al. 2023a, Zhang et al. 2023, Shi et al. 2024). This allows models to forecast sudden changes between successive positions that are physically impossible. Such forecasts resemble noisy versions of smooth ground truth trajectories. Therefore, lossy compression, which smooths trajectories, can improve modeling trajectories. Specifically, modeling compressed trajectory representations by appending the decompression operation to a learned model prevents the model from forecasting noisy trajectories. Furthermore, compressed representations reduce modeling complexity since the output space is reduced to the number of compressed parameters.

Related methods compress trajectories using principal component analysis (Jiang et al. 2023), discrete cosine transforms (Mao et al. 2019), eigenvalue decomposition (Bae et al. 2023), or Bézier curves (Hug et al. 2020).

We use discrete cosine transforms to compress trajectory representations, see Chapter 5. In contrast to Mao et al. (2019), we compress probabilistic trajectories representations. Unlike Jiang et al. (2023), Bae et al. (2023), our method is not data dependent, making it more generalizable.

[6] Also referred to as symmetric generalized normal distributions.

2.2 Representation learning

In this section, we describe fundamentals and related work in representation learning (Bengio et al. 2013). Representation learning is a branch of artificial intelligence research also referred to as deep learning. We first cover self-supervised learning methods, which learn good representations using self-generated training targets. Afterwards, we introduce mechanistic interpretability methods that reverse-engineer learned mechanisms and representations. Then, we describe sparse dictionary learning with sparse autoencoders. Furthermore, we cover the phenomenon of neural collapse in representation learning and introduce ensemble learning. Finally, we describe transformer models.

2.2.1 Self-supervised learning

The goal of self-supervised learning is to learn good representations using self-generated training targets. Self-supervised learning is typically used as pre-training before supervised fine-tuning. In this way, it can provide better initialization to improve downstream performance, or to reduce the amount of samples required to learn the downstream task. In general, research in self-supervised learning is roughly divided into generative and discriminative methods.

2.2.1.1 Generative self-supervised learning

Common generative self-supervised learning objectives are to sequentially auto-complete data or to reconstruct randomly masked data. For example, Van Den Oord et al. (2016), Chen et al. (2020a), Ren et al. (2024) train auto-regressive models to auto-complete images pixel by pixel. In natural language processing, such auto-regressive pre-training led to the success of GPT models (Radford et al. 2018, Brown et al. 2020). Similarly, masked autoencoding reconstructs randomly masked data and is successfully applied in language modeling (Devlin et al. 2019,

Warner et al. 2024) and computer vision (He et al. 2022, Feichtenhofer et al. 2022).

In motion forecasting, related methods (Cheng et al. 2023, Chen et al. 2023, Yang et al. 2023, Lan et al. 2024) use masked autoencoding to learn good representations by reconstructing masked map and trajectory data. The random masking that generates reconstruction targets is performed without human intervention, but the masked modalities like trajectories and map polylines are largely based on human annotations. Therefore, these methods rely more on human intervention than masked autoencoding in computer vision, where the reconstruction targets are randomly selected image (He et al. 2022) or video patches (Feichtenhofer et al. 2022). However, the level of human intervention in pre-training for motion forecasting is comparable to the level in language modeling (Radford et al. 2018, Devlin et al. 2019), where the token targets are based on text written by humans.

2.2.1.2 Discriminative self-supervised learning

The goal of discriminative self-supervised learning is to learn well-separated representations. Well-separated representations reside in a latent space, where representations of semantically similar samples are close to each other and far from representations with different semantics.

Contrastive learning (Chopra et al. 2005, Chen et al. 2020b, He et al. 2020) maximizes the similarity of representations of positive examples and minimizes the similarity to negative examples. Positive examples are differently augmented views of the same sample. Negative examples are views of randomly selected other samples. Oord et al. (2018) combine contrastive learning with auto-regressive modeling. Radford et al. (2021) propose contrastive language-image pre-training (CLIP), which learns from positive and negative examples of matching and mismatching language and image representations.

Related methods do not explicitly sample negative examples, but use momentum encoders (Grill et al. 2020), distillation (Caron et al. 2021), or auxiliary regularization (Zbontar et al. 2021, Bardes et al. 2022) to avoid representation collapse.

Representation collapse refers to the undesirable phenomenon that learned representations collapse to trivial[7] or redundant vectors (Bardes et al. 2022, Barbero et al. 2024).

For motion forecasting, Xu et al. (2022) use contrastive learning to learn well-separated representations of trajectory and map data. They maximize the similarity of trajectory and map representations from the same traffic scene, while minimizing the similarity to representations from other scenes.

We propose a marginal motion forecasting model that learns good representations via two types of redundancy reduction, (1) architecture-induced and (2) self-supervised redundancy reduction, see Section 3.1. Furthermore, we introduce a self-supervised pre-training objective specifically designed for joint motion forecasting of multiple agents, see Section 3.2. In contrast to Xu et al. (2022), our objectives do not explicitly sample negative examples, which are difficult to define for map and scene-wide trajectory data.

2.2.2 Mechanistic interpretability

Research on mechanistic interpretability analyzes and interprets how learned models work. Particularly, learned mechanisms and representations are studied (Meng et al. 2022, Chughtai et al. 2023, Nanda et al. 2023, Kitouni et al. 2024). Mechanistic interpretability methods do not enforce interpretability by handcrafting certain rules, but instead attempt to reverse-engineer learned mechanisms. For example, Meng et al. (2022) find that facts are stored in middle-layer feed-forward modules of language models and modify learned facts through causal interventions. Chughtai et al. (2023) show that different models learn similar features and mechanisms for group operations. Nanda et al. (2023) reverse-engineer that transformer models learn Fourier transforms to solve addition tasks with rotations. Furthermore, they reveal that grokking (Power et al. 2022) arises from learning

[7] Contrastive learning without negative examples leads to identical representations for all samples, like zero vectors (cf. Bardes et al. 2022).

structured mechanisms that replace initial memorization components. Kitouni et al. (2024) find that neural networks trained on nuclear physics content learn useful representations that align with human knowledge.

We find the vector representation of interpretable features (e.g., speed, direction, or agent type) learned by motion forecasting models and shows their impact on forecasts. Specifically, we modify learned representations at inference and analyze the resulting changes in motion forecasts, see Chapter 4. Furthermore, we reveal retrocausal mechanisms in forecasting models, where modifications to later parts of trajectories affect earlier parts as well, see Chapter 5.

2.2.3 Sparse dictionary learning

Representations learned by deep neural networks typically embed features across multiple dimensions. This is due to the phenomena of superposition, polysemanticity, or distributed representations (cf. Gurnee et al. 2023, A.3). Sparse dictionary learning[8] attempts to find a sparse representation of the input data to separate embedded features.

A recent method are sparse autoencoders (SAEs) (Bricken et al. 2023, Cunningham et al. 2023, Gao et al. 2025), which are used for interpretability research of language models. SAEs learn to encode and decode learned representations using sparse intermediate representations. The sparsity of intermediate representations is enforced with L_1-terms (Bricken et al. 2023, Cunningham et al. 2023) or top-k functions (Gao et al. 2025). In sparse intermediate representations, features are embedded in fewer or, ideally, one distinct dimension. Thus, sparse intermediate representations are a form of dictionaries, where each dimension is a dictionary entry. Templeton et al. (2024) show that learned dictionaries include interpretable features for concepts like the Golden Gate Bridge, code errors, secrecy, or discreetness.

[8] Also known as sparse coding (Olshausen and Field 1997, Lee et al. 2006).

We use SAEs to extract more distinct representations of interpretable features learned by forecasting models. Notably, this leads to more linear changes in forecasts when modifying these features at inference, see Chapter 4.

2.2.4 Neural collapse

Papyan et al. (2020) introduce the term neural collapse to refer to a desirable phenomenon in representation learning for classification. It describes that top-layer representations form semantic clusters, which collapse to their cluster means in the later epochs of training. Moreover, during training, the cluster means become approximately equidistant and equiangular vectors when centered at the global mean[9]. This means that the norm of all mean vectors and the angles between them become almost identical. Therefore, neural collapse describes a distinct form of well-separated representations, where all classes are linearly separable.

Later works show that neural collapse occurs in transfer learning (Galanti et al. 2021), self-supervised learning (Ben-Shaul et al. 2023), language modeling (Wu and Papyan 2024), and regression (Andriopoulos et al. 2024) as well.

We measure neural collapse to show that interpretable features, like speed or agent type, are separable in learned representations of forecasting models, see Chapter 4. Moreover, we use metrics for neural regression collapse to estimate the true dimensionality of learned trajectory representations, see Chapter 5.

2.2.5 Ensemble learning

Ensemble learning improves predictive performance by combining multiple base models (Ganaie et al. 2022). Base models are typically trained with different strategies or on different dataset splits and may have different model architectures.

[9] Also known as simplex equiangular tight frame (Zhu et al. 2021, Markou et al. 2024).

The most common methods for ensemble learning are bagging, boosting, and stacking.

Bagging trains base models independently on randomly sampled subsets of the training data. At inference, the predictions of all base models are aggregated by averaging for regression tasks or majority voting for classification tasks. This reduces variance and prevents overfitting since such ensembles are less sensitive to data noise.

Boosting trains base models sequentially to address the errors made by previous based models in an iterative manner. Specifically, later base models process the inputs and the predictions of earlier base models. At inference, the last base model's predictions or an aggregated form of multiple predictions are used. This reduces bias by focusing the training on challenging samples.

Stacking involves training multiple base models, which may have different architectures, on the same data. Then, a meta-learner, typically a small neural network, learns to best combine the base models' predictions. This method harnesses the strengths of different models to improve accuracy and reduce variance and bias.

Sparse mixture of experts (SMoE) is a form of ensemble learning that is popular in modern transformer models (Fedus et al. 2022). In case of SMoE methods, base models or sub-networks are referred to as experts. SMoEs are most closely related to stacking methods, but introduce input-dependent routing that determines which expert(s) are used per sample or per token. Early SMoE methods (Jacobs et al. 1991) train entire neural networks as experts, while more recent methods use layer-wise experts (Eigen et al. 2013, Jiang et al. 2024).

For motion forecasting, an ensembling method similar to bagging and stacking is typically used to improve benchmark performance (cf. Chai et al. 2020, Shi et al. 2022, Zhou et al. 2023a, Seff et al. 2023). These methods train multiple base models independently with identical model architectures but different initialization seeds. At inference, the trajectory predictions are combined by clustering the trajectory endpoints and afterwards averaging the trajectories within a cluster

(i.e., aggregation) or selecting the trajectory with the highest predicted probability per cluster (i.e., non-maximum suppression).

We use a form of SMoE to improve benchmark performance and train three expert models (see Section 5.2.1.3). At inference, we use a rule based router that selects one expert model based on the agent type. This reduces the number of active parameters per sample and removes the need for clustering to combine multiple predictions.

2.2.6 Transformer models

The transformer model (Vaswani et al. 2017) is a deep neural network that uses attention mechanisms and feed-forward layers to process sequential data. While it was originally designed for language modeling, the transformer model has since become the foundation for modern deep learning tasks, including computer vision (Dosovitskiy et al. 2021), reinforcement learning (Chen et al. 2021b), audio modeling (Radford et al. 2023), and robotics (Kim et al. 2024).

Information generally flows through a transformer model as follows. Input data is split into discrete tokens, which are then converted into dense vectors (i.e., embeddings) by embedding layers. Afterwards, a positional encoding (e.g., a sinosoidal (Vaswani et al. 2017) or a rotary encoding (Su et al. 2024)) or a learned positional embedding (Radford et al. 2018, Devlin et al. 2019) is added. Then, attention mechanisms enable the model to focus on the most relevant parts of the embeddings, by dynamically assigning attention weights as needed for the current prediction. Specifically, the model learns three linear projections of embeddings, named queries $\boldsymbol{Q}$, keys $\boldsymbol{K}$, and values $\boldsymbol{V}$. The scaled dot-product attention operation computes attention weights between each query and key, and multiplies them with the values

$$\text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_{\text{key}}}}\right)\boldsymbol{V}, \tag{2.14}$$

where d_{key} is the dimension of key vectors.

Finally, feed-forward layers that form a feed-forward network (FFN) update the embedding vectors with further projections

$$\mathrm{FFN}(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2, \tag{2.15}$$

with rectified linear units (ReLU), weights $\boldsymbol{W}$, and biases $\boldsymbol{b}$.

Both the attention and feed-forward modules are wrapped with residual connections and layer normalization, stabilizing training and improving the gradient flow. These modules are repeated multiple times within transformer models, allowing each subsequent module to focus on different information.

Furthermore, a standard transformer consists of two main components: an encoder and a decoder. The transformer encoder processes all input tokens in parallel using the aforementioned mechanisms. The transformer decoder generates new output tokens auto-regressively using the embeddings of the input tokens as context. Thus, the decoder performs two types of attention operations. The first is regular self-attention between embeddings of previously generated tokens, and the second is cross-attention between embeddings of new tokens and embeddings of input tokens. Specifically, the $\boldsymbol{Q}$ matrices are generated from embeddings of new tokens, while the $\boldsymbol{K}$ and $\boldsymbol{V}$ matrices are generated from embeddings of input tokens.

More recent transformer models are encoder-only (e.g., Devlin et al. 2019, Dosovitskiy et al. 2021) or decoder-only models (e.g., Radford et al. 2018, Brown et al. 2020, Jiang et al. 2024).

For motion forecasting, the majority of recent methods is transformer-based as well (e.g., Ngiam et al. 2022, Shi et al. 2022, Nayakanti et al. 2023, Zhou et al. 2023a, Seff et al. 2023, Zhang et al. 2023, Zhou et al. 2023b, Shi et al. 2024). These forecasting models learn from multimodal data, which includes tokenized input representations of trajectory, map, and traffic light data. While these methods achieve state-of-the art performance in terms of forecasting metrics, the authors only perform ablations on architectural choices.

Thus, it largely remains unknown, how these models learn to perform motion forecasting or represent interpretable features internally. Therefore, we propose methods that provide pre-training settings in which interpretable mechanisms are more likely to be learned (see Chapter 3) and reveal the learned representation of interpretable features (see Section 4.1) and learned mechanisms (see Section 4.2 and Section 5.2.3).

3 Self-supervised pre-training with interpretable objectives

By design, self-supervised learning excels in applications with large amounts of unlabeled data and limited labeled data (Chen et al. 2020b, Grill et al. 2020, Brown et al. 2020). Nevertheless, recent self-supervised methods combined with supervised fine-tuning can outperform plain supervised learning when using the same data (Caron et al. 2021, He et al. 2022) and require shorter overall training times (Feichtenhofer et al. 2022). This makes self-supervised pre-training a versatile choice for improving existing methods in a wide range of applications.

This chapter covers two methods for self-supervised pre-training for motion forecasting. The first, RedMotion, uses the redundancy reduction principle to learn good representations of the environment in traffic scenarios. RedMotion generates fixed-size embeddings that are invariant to moderate augmentations of variable-size inputs, see Section 3.1. The second, JointMotion, extends masked autoencoding by learning joint global representations from inputs of different modalities. Specifically, the similarity of embeddings of motion and map data is maximized as pre-training, see Section 3.2.

With both methods, we specifically pre-train transformer models (see Section 2.2.6) for two reasons: Unlike convolutional neural networks, transformer models have no inductive biases[1] for learning representations based on spatial spatial correlations (cf. Raghu et al. 2021). Therefore, such mechanisms must be learned from data. Furthermore, the pre-training followed by fine-tuning

[1] Like translation equivariance, where shifts in input images shift learned representations in the same manner.

paradigm is successfully used to train transformer models in other domains (e.g., Radford et al. 2018, He et al. 2022).

3.1 Self-supervised redundancy reduction for motion forecasting

In similar environments, such as two four-way stops with road users at similar locations, motion tends to be similar as well. Therefore, we propose to maximize the similarity of representations for similar environments as pre-training for motion forecasting.

We generate similar environments in a self-supervised manner by randomly augmenting a given environment. As augmentations, we apply moderate shift and rotation operations to map and agent data. Our method then learns to generate fixed-size embeddings that are invariant to such moderate augmentations. This includes two types of redundancy reduction mechanisms, (1) implicit architecture-induced redundancy reduction and (2) explicit redundancy reduction as pre-training objective. The architecture-induced redundancy reduction mechanism reduces information from variable-size inputs of agent and map data to a smaller fixed-size embedding, see Section 3.1.1. The redundancy reduction-based pre-training objective further reduces the redundancy of embedding components. This ensures that while maximizing the similarity of embeddings from similar environments, the embeddings do not collapse to trivial solutions like zero vectors, see Section 3.1.2.

To reflect the underlying **red**undancy **red**uction mechanisms, we refer to the proposed transformer model as RedMotion. Figure 3.1 shows the model architecture including the two redundancy reduction mechanisms.

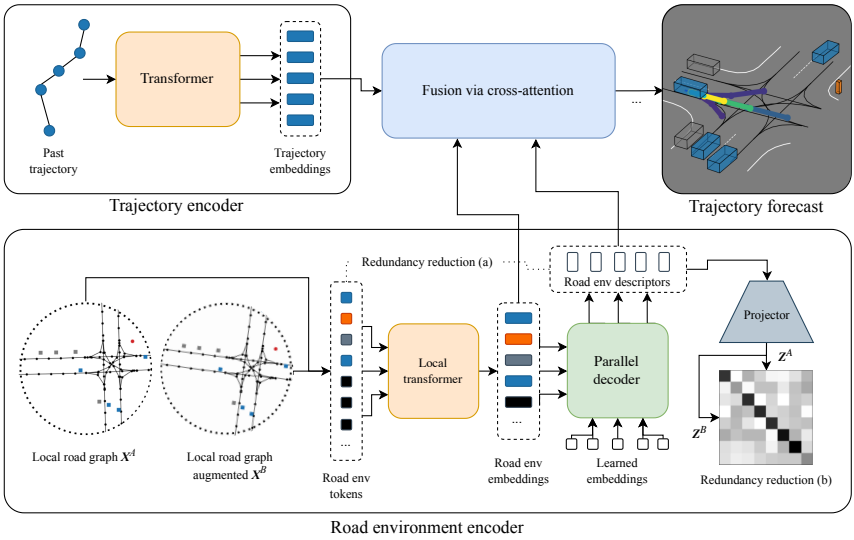


Figure 3.1: RedMotion. Our model consists of two encoders. The trajectory encoder generates an embedding for the past trajectory of the current agent. The road environment encoder generates fixed-size environment embeddings as context. We use two redundancy reduction mechanisms to learn good representations of road environments, (a) implicit see Section 3.1.1 and (b) explicit see Section 3.1.2. All embeddings are fused via cross-attention. Finally, a decoder generates a distribution of future trajectories per agent. Adapted from Wagner et al. (2024).

3.1.1 Architecture-induced redundancy reduction

The environment in traffic scenarios, which includes surrounding road users and map data, varies in complexity. Therefore, transformer models (e.g., Nayakanti et al. 2023, Zhang et al. 2023) or GNNs (e.g., Gao et al. 2020, Cui et al. 2023) for motion forecasting process variable-size vector representations of the environment, where more complex environments are represented with more vectors. For batch processing on GPUs, a fixed number of vectors is typically used as input with a Boolean mask that masks invalid placeholders in less complex scenarios.

However, our pre-training objective described in Section 3.1.2 requires fixed-size embeddings without masked elements.[2] Therefore, we propose an architecture-induced redundancy reduction mechanism that compresses information from variable-size inputs into smaller fixed-size representations.

As shown in Figure 3.1, the road environment encoder includes local attention mechanisms (Beltagy et al. 2020) to exchange information between the input vectors (i.e., road env tokens). The corresponding local transformer contains sequential blocks of local multi-head attention, LayerNorm, and MLP layers. Additionally, the inputs of each block are added to the normalized attention outputs and to the outputs of the MLP layers via residual connections.

The resulting road environment embeddings are then compressed into a fixed-size representation using a regular cross-attention mechanism. In this cross-attention mechanism, the queries are learned as fixed-size set of embedding vectors and the keys and values are learned as projections of the variable-size road environment embeddings. Thus, the output of this module (i.e., the road **env**ironment descriptors) shares the size with the learned queries.

3.1.2 Redundancy reduction as pre-training objective

As mentioned in Section 2.2.1.2, discriminative self-supervised learning with contrastive objectives (Chen et al. 2020b, Radford et al. 2021) can be counterproductive. Specifically, when randomly selected negative examples are similar to the positive examples. An example in our application is when the positive examples are augmented versions of a protected right-turn and the negative examples include protected right-turns as well. Supervised contrastive learning (Khosla et al. 2020) mitigates this issue by selecting negative examples that have different class labels than the positive examples. However, defining class labels in our

[2] While it is technically possible to mask out elements in the cross-correlation matrix as well (cf. Equation (3.1)), this would lead to less balanced training. In particular, embedding elements at higher indices would be included less often in the gradient computation, since they would be masked for the majority of less complex scenarios.

application to select negative examples would require non-trivial heuristics and ablations to answer questions like: *Is a four-way stop a good negative example for a signalized intersection?*

Therefore, we propose a self-supervised learning method that does not require negative examples. Following Zbontar et al. (2021), we use the redundancy reduction principle[3] to maximized the similarity of positive examples. Let $\boldsymbol{X}^A, \boldsymbol{X}^B$ be batches, where samples at the same batch index are positive examples generated from the same environment and two random augmentation sets A and B. We use continuous uniform distributions to randomly sample rotation augmentations in degrees with $\mathcal{U}(-10, 10)$ and shift augmentations in meters with $\mathcal{U}(-1, 1)$.

Afterwards, the road environment encoder generates fixed-size embeddings that represent the augmented environments (see road **env**ironment descriptors in Figure 3.1). We follow Zbontar et al. (2021) and use an MLP-based projector to further increase the dimension of the embeddings during pre-training. The batches of projected embeddings are referred to as $\boldsymbol{Z}^A$ and $\boldsymbol{Z}^B$. As pre-training objective, we minimize the difference between the cross-correlation matrix $\boldsymbol{C}$ of the projected embeddings and the identity matrix of the same size. Formally, we minimize

$$\mathcal{L}(\boldsymbol{C}) = \lambda_{\text{red}} \sum_i \sum_{i \neq j} {C_{i,j}}^2 + \sum_i (1 - C_{i,i})^2, \tag{3.1}$$

where i, j index the cross-correlation matrix and λ_{red} is a tunable weighting factor. The first term reduces redundancy by minimizing the correlation between embedding elements at different indices (off-diagonal elements of $\boldsymbol{C}$). The second term maximizes the similarity of the two embeddings by maximizing the correlation of embedding elements at the same index in both embeddings (on-diagonal elements of $\boldsymbol{C}$). Notably, this objective does not require negative examples since the redundancy reduction term ensures that embeddings do not collapse to trivial representations like zero-vectors.

[3] Barlow (2001) introduced the term in neuroscience.

3.1.3 Comparison with self-supervised pre-training methods focusing on marginal forecasting

We compare our self-supervised pre-training method with related discriminative and generative pre-training methods. Specifically, we compare our method with contrastive learning via PreTraM (Xu et al. 2022), self-distillation via GraphDINO (Weis et al. 2023), and masked autoencoding via Traj-MAE (Chen et al. 2023).

Since self-supervised pre-training for motion forecasting is only recently being developed, there are no common baseline models. Therefore, we use a modified version of our proposed model as baseline. The architecture-induced redundancy reduction mechanism (see Section 3.1.1) is our contribution, hence we remove the parallel decoder in the road environment encoder for the baseline version (see Figure 3.1). To give both models a similar capacity, we increase the hidden dimension for the baseline from 128 to 192, resulting in a RedMotion baseline with 9.9M trainable parameters and a RedMotion model with 9.2M parameters.

3.1.3.1 Dataset splits

We use the official training and validation splits of the Waymo and the AV2F dataset (see Section 2.1.5) as training and validation data. Since pre-training is particularly useful when little annotated data is available, we use 100% of the training data for pre-training and fine-tune on only 12.5%, following common practice in self-supervised learning (cf. Balestriero et al. 2023). In addition to the road environment, we use the trajectories of all traffic agents from the last 1.1 seconds (Waymo) and last 5 seconds (AV2F) as input during fine-tuning for motion forecasting. The datasets are sampled with 10 Hz, accordingly we use 11 and 50 past time steps as input.

3.1.3.2 Evaluation metrics

We use the L5Kit (Houston et al. 2021) to compute displacement errors as forecasting metrics (see Section 2.1.4). For the Waymo dataset, minADE and minFDE metrics are computed at different prediction horizons of $3\,\mathrm{s}$ and $6\,\mathrm{s}$, and averaged. For the AV2F dataset, minADE and minFDE metrics are computed for the prediction horizon of $6\,\mathrm{s}$.

3.1.3.3 Experimental setup

For PreTraM, GraphDINO, and our pre-training method, we use the same augmentations described in Section 3.1.2. For Traj-MAE pre-training, we mask 60% of the road environment tokens and train to reconstruct them. For all methods, we train the pre-training objectives using our local road environment tokens, so that the lane network and social context are included. For PreTraM, we evaluate two configurations, map contrastive learning (MCL) and trajectory-map contrastive learning (TMCL). For MCL, the similarity of augmented views of road environments is maximized. For TMCL, the similarity of embeddings from road environments and past agent trajectories of the same scene is maximized.

We evaluate our method with four different configurations of redundancy reduction: with mean feature aggregation (mean-ag), with learned feature aggregation (learned-ag), with reconstruction (red-mae), and between environment and past trajectory embeddings (env-traj). Mean-ag refers to using the mean of the road env descriptor tokens as input to the projector in Figure 3.1. For learned-ag, we use an additional transformer encoder layer to reduce the dimension of road env descriptor tokens to 16 and concatenate them as input for the reduction projector. The red-mae configuration is inspired by masked sequence modeling and a form of redundancy reduction via reconstruction. In detail, we generate two views ($\boldsymbol{X}^A$ and $\boldsymbol{X}^B$) of road environments, randomly mask 60% of their tokens, and reconstruct $\boldsymbol{X}^A$ from the masked version of $\boldsymbol{X}^B$ and vice versa. Since we reconstruct cross-wise, the similarity between embedding representations of the augmented views is maximized during pre-training. The env-traj configuration

is inspired by TMCL and reduces the redundancy between embeddings of past agent trajectories and road env descriptor tokens. Therefore, this configuration is inherently cross-modal but requires annotations of past agent trajectories.

For pre-training and fine-tuning, we use AdamW (Loshchilov and Hutter 2019) as the optimizer. The initial learning rate is set to 10^{-4} and reduced to 10^{-6} using a cosine annealing learning rate scheduler (Loshchilov and Hutter 2016). We pre-train and fine-tune all configurations for 4 hours and 8 hours using data-parallel training on 4 A100 GPUs. Following Konev et al. (2022), we minimize the negative multivariate log-likelihood loss for fine-tuning on motion prediction.

3.1.3.4 Results

Table 3.1 shows the results of this experiment. Overall, all pre-training methods improve the prediction accuracy in terms of minFDE and minADE. For our baseline model, our explicit redundancy reduction reduction mechanism (see Section 3.1.2) ranks second for the minFDE metric, marginally behind Traj-MAE and PreTraM in its TMCL configuration (only 0.3% worse). In terms of minADE, our mechanism ranks third behind Traj-MAE and PreTraM-TMCL. However, our mechanism is much less complex and has less data requirements. Compared to Traj-MAE, no random masking and no complex reconstruction decoder (transformer model) are required. Compared to PreTraM-TMCL, no past trajectory data is required.

When comparing to methods with similar requirements, our method outperforms PreTraM-MCL (-8.8% vs. -15.5% in minFDE) and GraphDINO (-8.3% vs. -15.5% in minFDE). For PreTraM-MCL, the question arises: *What is a good negative road environment?* Road environments of agents close to each other (e.g., a group of pedestrians) are much more similar than, for example, images of different classes in ImageNet (e.g., of cars and birds). During self-supervised contrastive pre-training, all samples in a batch other than the current one are treated as negative examples. Therefore, the pre-training objective becomes to learn dissimilar embeddings for rather similar samples.

Dataset	Model	Pre-training	Config	minFDE ↓		minADE ↓	
Waymo	Baseline	None		1.371 ± 0.018		0.670 ± 0.009	
		Traj-MAE* (Chen et al. 2023)		1.154 ± 0.002	**-15.8%**	0.542 ± 0.001	-19.1%
		PreTraM (Xu et al. 2022)	MCL	1.250 ± 0.012	-8.8%	0.576 ± 0.004	-14.0%
		PreTraM (Xu et al. 2022)	TMCL*	1.154 ± 0.001	**-15.8%**	0.525 ± 0.001	**-21.6%**
		GraphDINO (Weis et al. 2023)		1.257 ± 0.010	-8.3%	0.586 ± 0.003	-12.5%
		ours	mean-ag	1.159 ± 0.006	-15.5%	0.557 ± 0.002	-16.9%
	RedMotion	ours	mean-ag	1.111 ± 0.002	-19.0%	0.568 ± 0.001	-15.2%
		ours	learned-ag	1.098 ± 0.001	-19.9%	0.555 ± 0.001	-17.2%
		ours	red-mae	1.092 ± 0.002	-20.4%	0.557 ± 0.001	-16.9%
		ours	env-traj*	1.058 ± 0.009	**-22.8%**	0.529 ± 0.004	**-21.0%**
AV2F	RedMotion	None		2.353 ± 0.024		1.157 ± 0.004	
		ours	mean-ag	2.285 ± 0.010	-2.9%	1.140 ± 0.003	-1.5%
		ours	env-traj*	2.265 ± 0.020	**-3.7%**	1.106 ± 0.008	**-4.4%**

Table 3.1: Comparing pre-training methods for motion prediction. Best scores are **bold**, second best are underlined. We evaluate on the Waymo Open Motion (Waymo) and the Argoverse 2 Forecasting (AV2F) datasets. We report mean ± standard deviation of 3 training runs per method and configuration. All methods are pre-trained on 100% and fine-tuned on 12.5% of the training sets. *Denotes methods that require past trajectory annotations. Adapted from Wagner et al. (2024).

For GraphDINO, we hypothesize that more hyperparameter tuning could further improve the performance (e.g., loss temperatures or teacher weight update decay).

When we combine our two redundancy reduction mechanisms a and b (lower group in Table 1), our RedMotion model outperforms all related methods by at least 4% in minFDE and achieves similar performance in the minADE metric. We hypothesize that the reason for the comparable worse performance in the minADE score is our trajectory decoding mechanism. Our MLP-based motion head regresses all points in a trajectory at once, thus individual points in a predicted trajectory are less dependent on each other than in recurrent decoding mechanisms. When fine-tuning, the error for the final trajectory point is likely to be higher than for earlier points and our model can learn to focus more on minimizing this loss term. Therefore, if pre-training improves the learning behavior of our model, this will affect the minFDE score more.

When comparing different configurations of our combined redundancy reduction objective (middle block in Table 3.1), the env-traj configuration performs best

and the mean-ag configuration performs worst. However, similar to PreTraM-TMCL our env-traj configuration learns to map corresponding environment embeddings and past trajectory embeddings close to each other in a shared embedding space. Therefore, past trajectory data is required, which makes this objective less self-supervised and rather an improvement in data (utilization) efficiency. The learned-ag and red-mae configurations perform both better than the mean-ag configuration (1% improvement in minFDE and 2% improvement in minADE) and rather similar to each other. Since the learned-ag configuration has a lower computational complexity (no transformer-based reconstruction decoder but a simple MLP projector), we choose this pre-training configuration in the following.

On the AV2F dataset (lower block in Table 3.1), we compare our RedMotion model without pre-training versus with our mean-ag and env-traj pre-training configurations. Our mean-ag configurations improves the minFDE score by 2.9% and our env-traj configurations by 3.7%. This shows that our pre-training methods improve forecasting metrics on both datasets. Overall, the achieved displacement errors are higher for the AV2F dataset than for the Waymo dataset, since the metrics are not averaged over multiple prediction horizons, and likely because the AV2F training split is smaller (about 1M vs. 2M agent-centric samples).

3.1.4 RedMotion as standalone model without pre-training

Our RedMotion model can also be trained without pre-training to allow a direct comparison with related work on motion forecasting that does not perform pre-training. In this section, we compare RedMotion with other recent models for marginal motion forecasting on the Waymo Motion Prediction Challenge. Following the evaluation protocol of Zhang et al. (2023), we compare our method against real-time capable methods[4] without extensive post-processing (e.g., MTR adv-ens

[4] The real-time capability of our model is shown in Appendix A.1.

aggregates 6 trajectories from an ensemble of 7 models with 64 trajectories for each agent per model, see Section 2.2.5).

As described in the previous section, our model with a basic MLP-based motion decoder (mlp-dec configuration) tends to focus more on later than on earlier trajectory points, which worsens minADE and mAP scores. Therefore, we additionally train a version of our model with a transformer decoder (tra-dec) as motion decoder, which is common among recent related methods (e.g., Girgis et al. 2022, Nayakanti et al. 2023, Zhang et al. 2023).

In detail, we use a decoder with learned query tokens, which are transformed into trajectory proposals via attending to fused trajectory and road environment embeddings. For both variants, we use our architecture-induced redundancy reduction mechanism to learn road environment embeddings. As embedding aggregation method, we use the learned-ag configuration from Section 3.1.3.3.

We use 100% of the Waymo Open Motion training set for training to compare the performance of our model with that of other recent models. We perform evaluation on the validation and test splits.

As post-processing, our method does not require trajectory aggregation (see Section 2.2.5). Thus, we follow Konev (2022) and modify only the predicted probabilities of similar trajectories to improve mAP scores.

3.1.4.1 Results

Table 3.2 shows the performance of our model in comparison to other motion forecasting models. The metrics with suffix "@8s" are computed for the whole prediction horizon of 8 s, the others are the average for the prediction horizons of 3 s, 5 s, and 8 s (as in the official Waymo benchmark).

On the validation split, our model with a basic MLP-based motion decoder achieves the second lowest minFDE@8s score. Our model with a transformer decoder as motion decoder achieves the second best scores for the minFDE, minADE, and minADE@8s metrics. This shows that a transformer decoder adds

modeling capacity and prevents our model from focusing too much on the final trajectory points during training.

On the test spilt, our model with a transformer decoder ranks second in terms of minADE and minADE@8s. For the main challenge metric, the Soft mAP score, our model outperforms all comparable models. For reference, methods that employ ensembling (see Section 2.2.5) achieve higher mAP scores yet comparable minADE and minFDE scores to our method. Therefore, they match our performance on average and are marginally better in assigning confidence scores, which likely stems from their extensive post-processing.

Split	Method	Config	minFDE ↓	minADE ↓	minFDE@8s ↓	minADE@8s ↓	Soft mAP ↑
Val	MotionCNN (Konev et al. 2022)	ResNet-18	1.640	0.815	-	-	-
	MotionCNN (Konev et al. 2022)	Xeption71	1.496	0.738	-	-	-
	MultiPath++ (Varadarajan et al. 2022)		-	-	2.305	0.978	-
	Scene Transformer (Ngiam et al. 2022)	marginal	1.220	0.613	2.070	0.970	-
	HPTR (Zhang et al. 2023)		**1.092**	**0.538**	**1.877**	**0.874**	-
	RedMotion (ours)	mlp-dec	1.271	0.701	<u>1.952</u>	1.110	-
	RedMotion (ours)	tra-dec	<u>1.137</u>	<u>0.550</u>	1.987	<u>0.901</u>	-
	MTR* (Shi et al. 2022)		1.225	0.605	-	-	-
	MTR++* (Shi et al. 2024)		1.199	0.591	-	-	-
Test	MotionCNN (Konev et al. 2022)	Xeption71	1.494	0.740	-	-	-
	Scene Transformer (Ngiam et al. 2022)	marginal	1.212	0.612	2.053	0.980	-
	HDGT (Jia et al. 2023)		**1.107**	0.768	**1.898**	1.284	0.371
	MPA (Konev 2022)		1.251	0.591	2.202	0.981	0.393
	HPTR (Zhang et al. 2023)		<u>1.139</u>	**0.557**	<u>1.954</u>	**0.910**	<u>0.397</u>
	RedMotion (ours)	tra-dec	1.165	<u>0.564</u>	2.024	<u>0.925</u>	**0.401**
	MTR* (Shi et al. 2022)		1.221	0.605	2.067	0.983	0.422
	MTR++* (Shi et al. 2024)		1.194	0.591	2.024	0.961	0.433
	Wayformer** (Nayakanti et al. 2023)	multi-axis	1.128	0.545	1.942	0.892	0.434
	MTR** (Shi et al. 2022)	adv-ens	1.134	0.564	1.917	0.915	0.459

Table 3.2: Comparing marginal motion forecasting models. The metrics with suffix "@8s" are computed for the whole prediction horizon of 8 s, the others are the average for the prediction horizons of 3 s, 5 s, and 8 s (as in the official Waymo benchmark). Best scores are **bold**, second best are <u>underlined</u>. *Denotes methods that require trajectory aggregation as post-processing. **Denotes methods that employ ensembling. Adapted from Wagner et al. (2024).

3.1.5 Qualitative results

Figure 3.2 shows marginal motion forecasts for a vehicle. Dynamic vehicles are marked as blue boxes, pedestrians as orange boxes, cyclists as green boxes, and static agents as grey boxes. Road markings are shown in white, traffic lane centerlines are black lines, and bike lane centerlines are red lines. The past trajectory of the ego agent is a dark blue line. The ground truth trajectory is cyan blue, the predicted trajectories are color-coded based on the associated probability score using the colormap on the left (unnormalized).

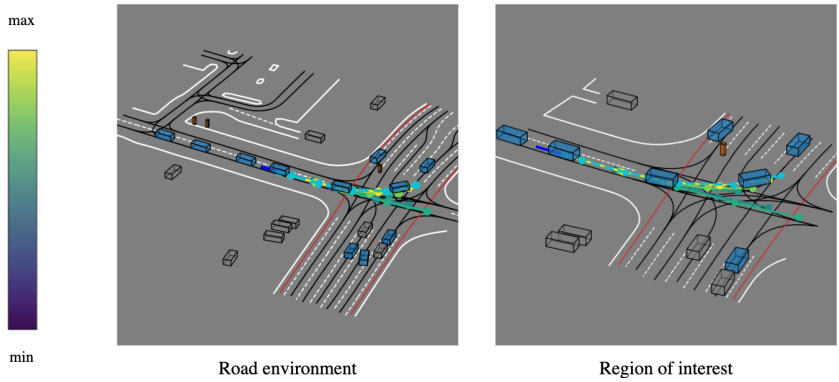


Figure 3.2: Vehicle motion forecasts. Dynamic vehicles are marked as blue boxes, pedestrians as orange boxes, cyclists as green boxes, and static agents as grey boxes. Road markings are shown in white, traffic lane centerlines are black lines, and bike lane centerlines are red lines. The past trajectory of the ego agent is a dark blue line. The ground truth trajectory is cyan blue, the predicted trajectories are color-coded based on the associated probability using the viridis colormap on the left. Adapted from Wagner et al. (2024).

Figure 3.3 shows motion forecasts for a cyclist. This plot shows an error case as the blueish trajectory pointing downwards enters the inbound lane. However, the predicted probability is fairly low.

Figure 3.4 shows motion forecasts for a pedestrian. Compared to the trajectories of vehicles and cyclists, the trajectories of pedestrians are more diffuse.

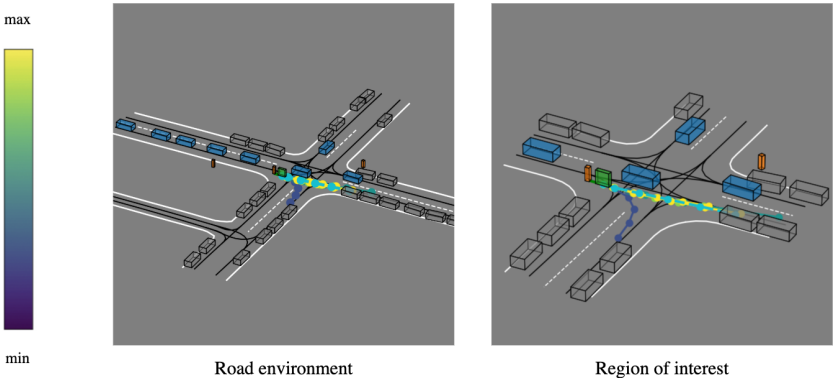


Figure 3.3: Cyclist motion forecasts. We use the same color-coding as in Figure 3.2. This plot shows an error case as the blueish trajectory pointing downwards enters the inbound lane. Adapted from Wagner et al. (2024).

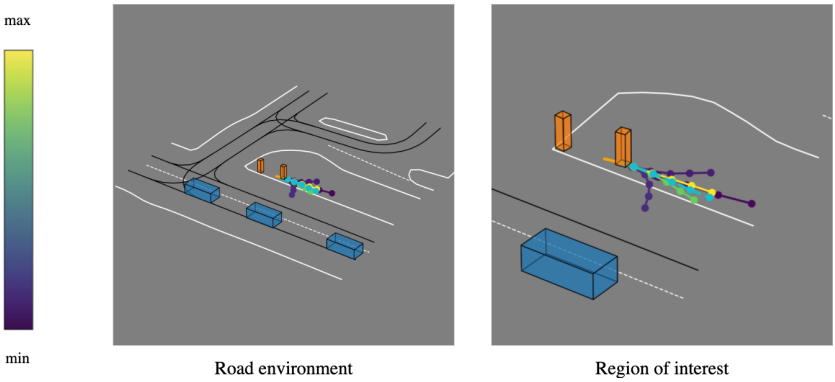


Figure 3.4: Pedestrian motion forecasts. We use the same color-coding as in Figure 3.2. Adapted from Wagner et al. (2024).

3.2 Multimodal self-supervised learning for joint motion forecasting

Marginal motion forecasting is a common and computationally efficient simplification (see Section 2.1.1). However, because it models agents individually in the output space, interaction modeling is limited. Therefore, in this section, we propose JointMotion, a method for pre-training joint forecasting models with improved interaction modeling.

Joint motion forecasting methods model motion for multiple agents jointly by predicting joint trajectory distributions (see Section 2.1.3). This requires global (i.e., scene-level) representations as context. Therefore, we propose learning scene-level representations by connecting motion and environment representations (see Figure 3.5 (a)). We complement this with an instance-level objective that reconstructs masked polylines of multiple modalities (see Figure 3.5 (b)).

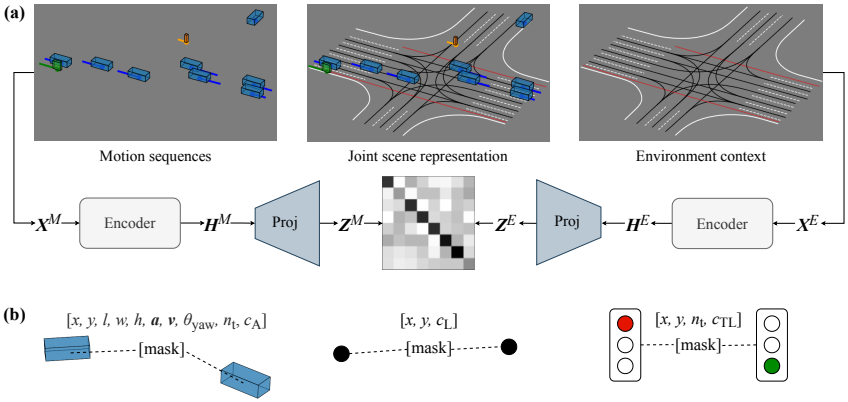


Figure 3.5: JointMotion. **(a)** Connecting motion and environments: Our scene-level objective learns joint scene representations via non-contrastive similarity learning of motion sequences M and environment context E. **(b)** Masked polyline modeling: Our instance-level objective refines learned representations via masked autoencoding of multimodal polyline embeddings (i.e., motion, lane, and traffic light embeddings). Adapted from Wagner et al. (2024a).

3.2.1 Connecting motion and environments

Motion forecasting models should learn which motion is likely in a given environment, including traffic rules and interaction among road users. We propose pre-training for this task by aligning motion and environment embeddings. This scene-level objective aims to learn a joint embedding space that connects motion and environment context. Consequently, models implicitly learn which motion fits a given environment and vice versa.

In detail, we use the past motion of all modeled agents to generate a scene-level motion embedding ($\boldsymbol{Z}_i^M$ in Figure 3.5, where $\boldsymbol{Z}^M$ is a batch of embeddings) and combine lane data and traffic light states to a corresponding environment embedding ($\boldsymbol{Z}_i^E$ in Figure 3.5). These embeddings are generated by modality-specific encoders (i.e., for motion, lane, and traffic light data) followed by global average pooling and an MLP-based projector (Proj in Figure 3.5). We perform average pooling on the intermediate embeddings ($\boldsymbol{H}_i^M$ and $\boldsymbol{H}_i^E$ in Figure 3.5) to be invariant to variations in the number of agents and the complexity of environments. Following Fini et al. (2023), we use two separate MLPs with LayerNorm and ReLU activation functions as projectors, each having a hidden dimension of 2048 and an output dimension of 256. We use the modality-specific encoders of recent motion prediction models (e.g., Ngiam et al. (2022), Zhang et al. (2023)) without modifications and remove the additional projector after pre-training.

The joint embedding space is learned by similarity learning via redundancy reduction. Following Zbontar et al. (2021), we reduce the redundancy of vector elements per embedding (i.e., for $\boldsymbol{Z}_i^M$ and $\boldsymbol{Z}_i^E$ individually) and maximize their similarity by approximating the cross-correlation matrix $\boldsymbol{C}$ of $\boldsymbol{Z}_i^M$ and $\boldsymbol{Z}_i^E$ to the identity matrix with the same shape:

$$\mathcal{L}_{\text{CME}}(\boldsymbol{C}) = \lambda_{\text{red}} \sum_i \sum_{j \neq i} {C_{ij}}^2 + \sum_i (1 - C_{ii})^2, \tag{3.2}$$

where i, j index the cross-correlation matrix. The redundancy reduction term is scaled by λ_{red} and ensures that individual embedding elements capture different features. Therefore, it prevents representation collapse with trivial solutions for all embeddings across all scenes (e.g., zero vectors, cf. Bardes et al. (2022)). Non-trivial yet identical solutions are not explicitly prevented, but are unlikely, as empirically shown by Zbontar et al. (2021).

Similar to the RedMotion pre-training (see Section 3.1.2), this objective is non-contrastive, which removes the need to define negative examples. Unlike the RedMotion pre-training, this objective maximizes the similarity of embeddings from different modalities. Specifically, embeddings of motion and environment context rather than embeddings of augmented views of the same environment, removing the requirement to develop suitable augmentations as well.

3.2.2 Masked polyline modeling

Scene-level representations are well suited to provide an overview of traffic scenes, but lack instance-level details. For example, the exact position of traffic agents or lane curvatures. Therefore, we combine our scene-level objective with the instance-level objective of masked polyline modeling (MPM) to refine learned representations.

Inspired by masked sequence modeling (Devlin et al. 2019, He et al. 2022), we mask elements of polylines representing past motion sequences, lanes, and past traffic light states and learn to reconstruct them from non-masked elements and environment context. As shown in Figure 3.5 (b), we represent traffic agents with 10 features rather than just past positions (as in Chen et al. (2023), Cheng et al. (2023)). In detail, we reconstruct agent positions (x, y), dimensions (l, w, h), acceleration $\boldsymbol{a}$, velocity $\boldsymbol{v}$, yaw angle θ_{yaw}, temporal indices n_{t}, and classes c_{A} (i.e., vehicle, cyclist, or pedestrian). For lanes, we reconstruct positions (x, y) and lane classes c_{L}. For traffic light state sequences, we reconstruct positions (x, y), temporal indices n_{t}, and state classes c_{TL} (i.e., green, yellow, or red). Following Zhang et al. (2023), we represent positions, dimensions, accelerations, velocities,

and yaw angles as float values and temporal order and agent classes as boolean one-hot encodings[5].

For models with late or hierarchical fusion mechanisms (e.g., Ngiam et al. (2022), Zhang et al. (2023)), we use the modality-specific encoders to generate embeddings per modality (see Figure 3.6 (a)). Afterwards, we concatenate these embeddings and use a shared local decoder to reconstruct masked sequence elements from non-masked elements and context from other modalities. As local decoder, we use transformer blocks with PreNorm (Nguyen and Salazar 2019), local attention (Beltagy et al. 2020) with an attention window of 32 tokens, 8 attention heads, rotary positional encodings (Su et al. 2024), and feed-forward layers with an input dimension of 256 and a hidden dimension of 1024.

For models that employ early fusion mechanisms (e.g., Nayakanti et al. (2023)), we use learned queries and a shared decoder to reconstruct the input sequences (see Figure 3.6 (b)). Such models learn a compressed latent representation for multi-modal input. Therefore, we use learned queries in the same number as input tokens to decompress these representations and reconstruct the input sequences. We use a regular cross-attention mechanism between the learned queries and compressed latent representations and a local self-attention mechanism within the set of learned queries. The resulting transformer blocks have the same structure and hyperparameters as in the late fusion setup described above.

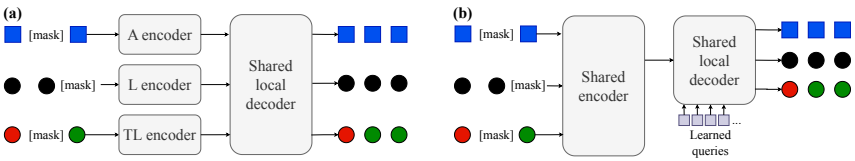


Figure 3.6: Adaptive decoding for masked polyline modeling with late and early fusion encoders. **(a)** Late fusion with modality-specific encoders for agents (A encoder), lanes (L), and traffic lights (TL). **(b)** Early fusion with a shared encoder for all modalities. Compressed features are decoded using learned query tokens. Adapted from Wagner et al. (2024a).

[5] For practical reasons, these are later converted to float arrays to concatenate them to the other features and efficiently process them on GPUs.

For both variants, we use random attention masks for masking and a masking ratio 60%. As training target, we minimize the Huber loss between the reconstructed polylines and the input polylines

$$\mathcal{L}_{\text{MPM}} = \lambda_{\text{A}}\mathcal{L}_{\text{A}} + \lambda_{\text{L}}\mathcal{L}_{\text{L}} + \lambda_{\text{TL}}\mathcal{L}_{\text{TL}}. \tag{3.3}$$

If not specified otherwise, we set $\lambda_{\text{A}} = \lambda_{\text{L}} = \lambda_{\text{TL}} = 1$.

3.2.3 Comparison with self-supervised pre-training methods focusing on joint forecasting

We compare our JointMotion method with recent self-supervised pre-training methods for motion forecasting using the Waymo Open Motion dataset (Ettinger et al. 2021). Specifically, we compare our method with contrastive learning via PreTraM (Xu et al. 2022) and masked autoencoding with Forecast-MAE (Cheng et al. 2023) and Traj-MAE (Chen et al. 2023).

3.2.3.1 Motion forecasting model

We pre-train and fine-tune the well-established Scene Transformer (Ngiam et al. 2022) model on the Waymo training split. We use the publicly available implementation by Zhang et al. (2023) with 3 modality-specific encoders (i.e., for agent, lane, and traffic light data). Adapted to the complexity per modality, we encode traffic light features with 1, agent features with 3, and lane features with 6 transformer blocks.

3.2.3.2 Pre-training

For all methods, we add a pre-training decoder for late fusion models (see Figure 3.6) with 3 transformer blocks and the hyperparameters described in Section 3.2.2. For our JointMotion method, we additionally add two projectors for

scene-level representations as described in Section 3.2.1. For PreTraM, we follow its trajectory-map contrastive learning configuration and add two linear projection layers as projectors for trajectory and map embeddings. For Forecast-MAE, we use a masking ratio of 60% and reconstruct positions of lane polylines and past trajectories by minimizing the MSE loss. We exclude future trajectories, since self-supervised learning by design does not use the same labels as the intended downstream task (cf. Balestriero et al. (2023)). For Traj-MAE, we use a masking ratio of 60% and reconstruct positions of lane polylines and past trajectories by minimizing the corresponding Huber loss. For our method, we minimize the joint loss of our proposed objectives

$$\mathcal{L}_{\text{JointMotion}} = \lambda_{\text{CME}}\mathcal{L}_{\text{CME}} + \mathcal{L}_{\text{MPM}}. \tag{3.4}$$

We set $\lambda_{\text{CME}} = 0.01$ and following Zbontar et al. (2021) the weight of the redundancy reduction term $\lambda_{\text{red}} = 0.005$.

3.2.3.3 Fine-tuning

For all methods, we replace the pre-training decoder with a shared global decoder and learned anchors for $K = 6$ future trajectories. We initialize the modality-specific encoders with the learned weights from pre-training and do not freeze any weights during fine-tuning. We fine-tune the Scene Transformer model using its joint configuration and hard loss assignment. Accordingly, the loss is computed for the best[6] scene-wide joint prediction mode. As post-processing, we follow Konev (2022) and adjust the predicted probabilities of redundant forecasts.

3.2.3.4 Training time, hardware, and optimizer

For all methods, we pre-train for 10 hours and fine-tune for 23.5 hours using a training server with 4 A100 GPUs. For pre-training and fine-tuning, we use

[6] Measured as lowest L_2-distance to the ground truth.

AdamW (Loshchilov and Hutter 2019) as optimizer with an initial learning rate of 1×10^{-4} and a step learning rate scheduler with a reduction rate of 0.5 and a step size of 25 epochs.

3.2.3.5 Results

Table 3.3 shows the results of this experiments. Explicit scene-level objectives (i.e., PreTraM (Xu et al. 2022) and our JointMotion) lead to better and more balanced performance across all agent types, while implicit scene-wide masked autoencoding with Forecast-MAE or Traj-MAE tends to focus more on the pedestrian class than on the others (see mAP scores). Therefore, it is likely that explict scene-wide objectives enforce learning interactions between varying agent types more.

Traj-MAE and our method without the objective of connecting motion and environments (JointMotion w/o CME) achieve better scores than Forecast-MAE. Consequently, reconstructing individual elements of polylines improves representations more than reconstructing whole polylines.

The superior performance of JointMotion w/o CME compared to Traj-MAE indicates that extending the motion feature set and reconstructing traffic light states as well further improves learned representations.

Our method without masked polyline modeling (JointMotion w/o MPM) performs on par with PreTraM, indicating that redundancy reduction can replace negative examples for scene-level similarity learning.

Overall, pre-training with both proposed objectives (i.e., last row in Table 3.3) leads to the best scores across all agent types. This shows that the combination of our objectives works best.

Figure 3.7 further highlights the complementary nature of our two objectives (CME and MPM). Specifically, the reconstruction of past traffic light states is learned very similar with both configurations. The reconstruction loss for past agent motion converges more slowly with JointMotion pre-training, but reaches

Pre-training	mAP↑				minADE↓			minFDE↓		
	avg	cyc	ped	veh	cyc	ped	veh	cyc	ped	veh
None	0.1596	0.1465	0.1821	0.1504	1.522	0.698	1.486	3.653	1.654	3.476
Forecast-MAE (Cheng et al. 2023)	0.1592	0.1420	0.1873	0.1482	1.529	0.694	1.423	3.664	1.652	3.343
Traj-MAE (Chen et al. 2023)	0.1677	0.1492	0.1917	0.1623	1.421	0.708	1.338	3.320	1.647	3.090
PreTraM (Xu et al. 2022)	0.1689	0.1724	0.1775	0.1569	1.488	0.711	1.461	3.489	1.657	3.374
JointMotion w/o MPM	0.1689	0.1762	0.1777	0.1528	1.478	0.684	1.406	3.507	1.608	3.287
JointMotion w/o CME	0.1784	0.1652	0.1903	0.1796	1.457	0.700	1.317	3.363	1.630	3.033
JointMotion	**0.1940**	**0.1970**	**0.1964**	**0.1886**	**1.343**	**0.677**	**1.288**	**3.095**	**1.583**	**2.941**

Table 3.3: Comparison of self-supervised pre-training methods for joint motion forecasting. All methods are used to pre-train Scene Transformer models (Ngiam et al. 2022) on the Waymo training split and are evaluated on the validation split. Agent types: cyclist (cyc), pedestrian (ped), and vehicle (veh). Best scores are **bold**, second best are underlined. Adapted from Wagner et al. (2024a).

similar values as well. However, with our scene-level objective (CME), the lane reconstruction loss likely converges to a higher value (see right plot in Figure 3.7). We hypothesize that models pre-trained with our scene-level objective tend to focus more on the overall lane structure than on specific details of individual lane polylines.

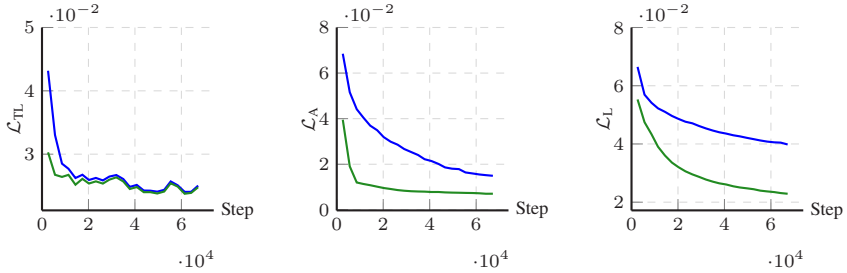


Figure 3.7: Loss plots of our complementary pre-training objectives. The blue curve represents JointMotion, while the green curve represents JointMotion w/o CME. L stands for lanes, TL stands for traffic lights, and A stands for agents. Adapted from Wagner et al. (2024a).

Figure 3.8 shows that Scene Transformer models pre-trained with our JointMotion method achieve higher mAP scores on the Waymo dataset than models trained from scratch, even in a shorter wall training time (pre-training + fine-tuning vs. training from scratch). This aligns with the finding by Feichtenhofer et al. (2022) that self-supervised pre-training can reduce overall training times.

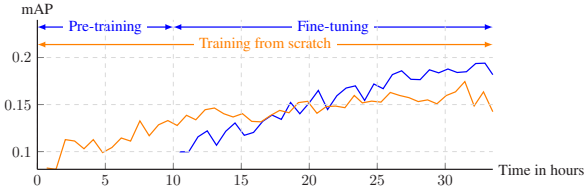


Figure 3.8: Accelerating and improving training via self-supervised pre-training. Scene Transformer models pre-trained with JointMotion achieve higher mAP scores on the Waymo dataset than models trained from scratch, even in a shorter total wall training time (pre-training + fine-tuning vs. training from scratch). Adapted from Wagner et al. (2024a).

3.2.4 Comparing scene-level pre-training methods

In this experiment, we further compare the scene-level objectives PreTraM (Xu et al. 2022) and JointMotion. We train Scene Transformer (Ngiam et al. 2022), HPTR (Zhang et al. 2023), and a joint configuration of Wayformer (Nayakanti et al. 2023) to cover all common types of environment representations in motion prediction (i.e., scene-centric, pairwise relative, and agent-centric).

3.2.4.1 Experimental setup

For Scene Transformer, we use the same configuration as in the previous experiment (see Section 3.2.3). For HPTR, we analogously add 3 modality-specific encoders and use a shared decoder for $K = 6$ trajectories per modeled agent. For the joint configuration of Wayformer, we follow Jiang et al. (2023) and use a shared encoder for early fusion, which compresses the multi-modal input to 128 embeddings. We concatenate multiple such agent-centric embeddings with positional and rotation information into a common reference frame and use a shared decoder to predict joint motion modes. For pre-training the Wayformer model, we add our decoder for early fusion configurations (see Figure 3.6). The Wayformer model is not pre-trainable with PreTraM since an instance-level objective is required to decode the modality-specific tokens from fused representations (cf. Section 3.2.2). For all models, we employ the same hardware, training time,

optimizer, and learning rate scheduling as in the previous experiment (see Section 3.2.3). We evaluate all configurations on the interactive validation split of the Waymo dataset and on the AV2F dataset (see Section 2.1.5).

3.2.4.2 Results

Table 3.4 shows the results of this experiment. Our JointMotion method consistently outperforms PreTraM using different models with varying environment representations and fusion mechanisms. Our method improves all models, while the improvement of the scene-centric Scene Transformer model is most significant (e.g., 12% lower minFDE) and the improvement of the agent-centric Wayformer model is least significant (e.g., 3% lower minFDE). Hence, the improvements are inversely proportional to the sample efficiency of the models. In scene-centric modeling, one sample is generated per scene, whereas in agent-centric modeling, one sample is generated for each modeled agent within a scene. This aligns with the finding that fine-tuning with more samples generally reduces the value of pre-training (Zoph et al. 2020).

Dataset	Model (config)	Pre-training	minFDE ↓		minADE ↓		MR ↓		OR ↓	
Waymo	Scene Transformer	None	3.6715		1.5255		0.7372		0.2868	
		PreTraM (Xu et al. 2022)	3.6508	-0.57%	1.5415	1.05%	0.7385	0.18%	0.2915	1.64%
		JointMotion	**3.2400**	-11.75%	**1.3830**	-9.36%	**0.7090**	-3.82%	**0.2847**	-0.73%
	HPTR	None	2.6003		1.1682		0.6030		0.2331	
		PreTraM (Xu et al. 2022)	2.5049	-3.66%	1.0981	-5.99%	0.5863	-2.78%	0.2345	0.60%
		JointMotion	**2.4006**	-7.68%	**1.0564**	-9.58%	**0.5591**	-7.28%	**0.2297**	-1.46%
	Wayformer (joint)	None	2.3529		1.0209		0.5461		0.2273	
		JointMotion	**2.2823**	-3.00%	**0.9939**	-2.64%	**0.5270**	-3.50%	**0.2143**	-5.72%
AV2F	HPTR	None	2.2550		1.1380		-		**0.0988**	
		JointMotion on Waymo	**2.1530**	-4.53%	**1.1370**	-0.09%	-		0.1025	3.75%

Table 3.4: Comparing scene-level pre-training methods. All metrics are computed using the Waymo Open Motion interactive (Waymo) and Argoverse 2 Forecasting (AV2F) validation splits. Best scores are **bold**. Adapted from Wagner et al. (2024a).

Unlike PreTraM, the proposed adaptive pre-training decoder (Figure 3.6) combined with our instance-level objective (Section 3.2.2) enable our method to adapt to models with early fusion mechanisms (e.g., Wayformer). Specifically,

modality-specific masked polyline modeling with learned queries in the same number as input tokens enables our method to decode modality-specific tokens (i.e., agent, lane, and traffic light tokens) from compressed latent representations. This is particular relevant since the current state-of-the-art methods[7] on the Argoverse 1 Forecasting (ProphNet (Wang et al. 2023b) via AiP tokens) and Argoverse 2 Forecasting (QCNeXt (Zhou et al. 2023b) via query-centric modeling) benchmarks rely on fusion mechanisms with compressed latent representations as well.

Furthermore, our pre-training leads to comparable improvements on the interactive and the regular validation splits (see Table 3.3), while pre-training with PreTraM leads to smaller improvements on the interactive validation split. We hypothesise that with our additional instance-level objective, more fine-grained trajectory details are learned, which is more important for close trajectories of interacting agents.

The lowest block in Table 3.4 shows that our method leads to transferable representations. Specifically, pre-training on the Waymo dataset improves fine-tuning on the AV2F dataset.

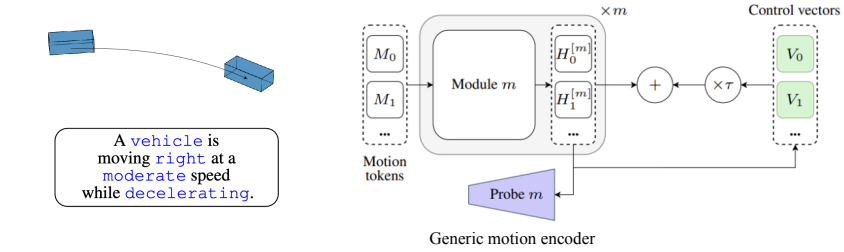
[7] As of Sep 27, 2024.

4 Interpretable control vectors for motion forecasting

Self-supervised pre-training with interpretable objectives is a step toward mechanistic interpretability of learned models. However, it does not reverse-engineer learned mechanisms; rather, it provides a training setting in which interpretable mechanisms are more likely to be learned.

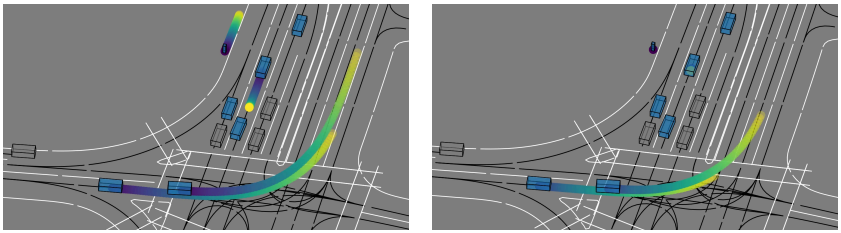
In this chapter, we introduce Words in Motion, a mechanistic interpretability method that reveals (1) the learned representations of interpretable features like speed or agent type and (2) how these features affect motion forecasts of transformer models.

Therefore, our method enables mechanistic interpretations and generalization to unseen dataset characteristics, like slower slower driving styles by modifying speed features (see Section 4.7).



(a) Interpretable categorical features

(b) Analyzing and modifying hidden states



(c) Modifying hidden states leads to sensible changes in motion forecasts

Figure 4.1: Words in Motion. **(a)** Inspired by words in natural language, we define categorical features with `interpretable states`. **(b)** We measure whether these features are embedded in the hidden states $\boldsymbol{H}$ of transformer models with linear probes. Furthermore, we use our categorical features to fit control vectors $\boldsymbol{V}_t$ that allow for modifying hidden states at inference. **(c)** Such modifications lead to sensible changes in motion forecasts, indicating that the analyzed features are functionally important.

4.1 Neural collapse toward interpretable features

Papyan et al. (2020) introduce the term *neural collapse* to describe that deep learning models often learn a distinct form of well-separated representations (see Section 2.2.4).

We use neural collapse as metric of interpretability and analyze representations learned by recent transformer models for motion forecasting. Precisely, we measure neural collapse toward interpretable features in learned motion representations (i.e., hidden states $\boldsymbol{H}$ in Figure 4.1 (b)). Following Ben-Shaul et al. (2023)[1], we measure neural collapse using linear probes (Alain and Bengio 2017). This focuses the analysis of neural collapse toward our categorical features on the aspect whether these features are linearly separable.

Drawing inspiration from words in natural language, we define categorical features with interpretable states. We define a direction feature and its states using the cumulative sum of differences in yaw angles, assigning it to either `left`, `straight`, or `right`. Additionally, we introduce a `stationary` state for stationary objects, where direction lacks semantic significance. We define further states for speed, dividing the speed values into four intervals: `high`, `moderate`, `low`, and `backwards`. Lastly, we analyze the change in acceleration by comparing the integral of speed over time to the projected displacement with initial speed. Accordingly, we classify acceleration profiles as either `accelerating`, `decelerating`, or `constant` (see Figure 4.1 (a)). All thresholds and further details are in Appendix A.2.

4.1.1 Experimental setup

4.1.1.1 Motion forecasting models

We analyze three recent transformer models for motion forecasting, Wayformer (Nayakanti et al. 2023), HPTR (Zhang et al. 2023) and our RedMotion (Section 3.1.1) model. Wayformer and our RedMotion models employ attention-based scene encoders to learn agent-centric embeddings of past motion, map, and traffic light data. To efficiently process long sequences, Wayformer uses latent query attention (Jaegle et al. 2021) for subsampling, RedMotion lowers memory

[1] Ben-Shaul et al. (2023) show that linear probing accuracy follows similar trends as the accuracy of nearest class-center classifiers (NCCs), which are typically used to measure neural collapse.

requirements via local-attention (Beltagy et al. 2020) and architecture-induced redundancy reduction. In this experiment, we do not pre-train RedMotion, but train it fully supervised as the other models. HPTR models learn pairwise-relative environment representations via kNN-based attention mechanisms. For Wayformer, we use the implementation by Zhang et al. (2023) and the early fusion configuration. Therefore, we analyze the hidden states generated by an MLP-based input projector for motion data, which consists of three layers. For RedMotion and HPTR, we use the publicly available implementations. We configure RedMotion with a late fusion encoder for motion data, and HPTR using a custom hierarchical fusion setup with a modality-specific encoder for past motion with a shared encoder for environment context. Further details on learned fusion mechanisms are in Appendix A.3.

4.1.1.2 Linear probes

We add linear probes for our categorical motion features (see Section 4.1) to hidden state of all models ($\boldsymbol{H}_{t,:}^{(m)}$ in Figure 4.1, where $m \in \{0, 1, 2\}$ is the module number and t is the temporal index). These one-layer classifiers are learned during training using regular cross-entropy loss to classify speed, acceleration, direction, and the agent classes from hidden states. Note that we detach this objective from the overall gradient computation and use a separate optimization loop for each linear probe. Therefore, these probes do not update hidden states, but measure whether our categorical features are becoming more linearly separable during regular training for motion forecasting.

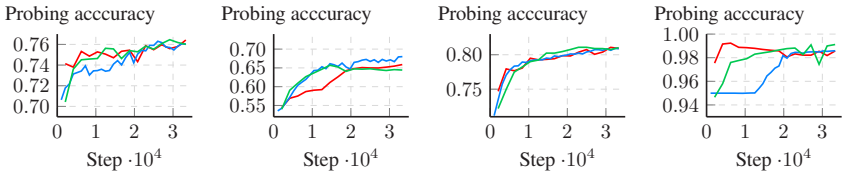
4.1.1.3 Training details and hyperparameters

We provide Wayformer and HPTR models with the nearest 512 map polylines, and RedMotion model with the nearest 128 map polylines. All models process a maximum of 48 surrounding traffic agents as environment context, have a hidden dimension of 128, and are configured to forecast $K = 6$ trajectories per agent. For the AV2F dataset, we use past motion trajectories of 50 time steps (representing

5 s) as input. For the Waymo dataset, we use past motion trajectories with 11 steps (representing 1.1 s) as input. For Wayformer and RedMotion, we use the unweighted sum of the negative log-likelihood loss for positions modeled as mixture of normal distributions and cross-entropy for probabilities as motion forecasting loss. For HPTR, we additionally use the cosine loss for the heading angle and the Huber loss for velocities. For all models, we use AdamW (Loshchilov and Hutter 2019) in its default configuration as optimizer and set the initial learning rate to 2×10^{-4}. As post-processing, we follow Konev (2022) and reduce the predicted probabilities of similar trajectories.

4.1.2 Experimental results

Figure 4.2 shows the linear probing accuracies for our interpretable motion features for the AV2F dataset. The scores are computed on the validation split over the course of training. All models achieve similar accuracy scores, while the Wayformer model achieves slightly higher scores for classifying acceleration and lower scores for agent classes. Overall, we measure high linear probing accuracy for all intepretable features. This shows that all models likely exhibit neural collapse toward our interpretable features.



(a) Speed **(b)** Acceleration **(c)** Direction **(d)** Agent type

Figure 4.2: Linear probing accuracies for RedMotion, Wayformer, and HPTR on the validation split of the AV2F dataset, for module 2. Adapted from Tas and Wagner (2025).

The representation quality metric normalized standard deviation of embeddings is shown in Figure 4.3, as defined by Chen and He (2021). Both HPTR and RedMotion learn to generate embeddings with a normalized standard deviation close to the desired value of $1/\sqrt{d}$ (cf. Chen and He 2021), where d is the hidden dimension. The scores for Wayformer are lower, which reflects differences between attention-based and MLP-based motion encoders.

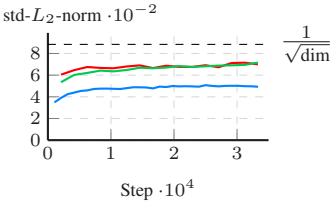
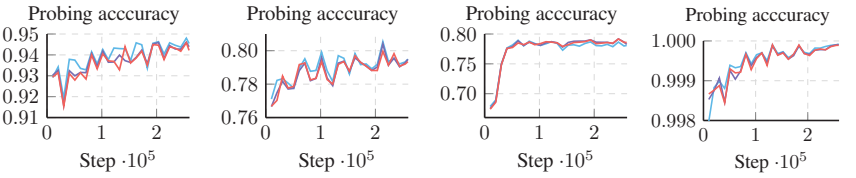


Figure 4.3: Normalized standard deviation of representations for RedMotion, Wayformer, and HPTR. Adapted from Tas and Wagner (2025).

Figure 4.4 shows the linear probing accuracies for our interpretable features on the Waymo dataset. Here, we report the scores for each of the three hidden states $\boldsymbol{H}^{(m)}$ in the RedMotion model (i.e., after each module m in the motion encoder, see Figure 4.1). Similar accuracy scores are reached for all features at all three hidden states. The accuracies for the speed and acceleration classes continuously improve, while those for direction classes reach 0.80 early on. Compared to the direction scores on the AV2F dataset, the scores on the Waymo dataset "jump" earlier. We hypothesize that this is linked to the shorter input motion sequence on Waymo ($1.1\,\mathrm{s}$ vs. $5\,\mathrm{s}$), which limits the amount possible movements. In contrast to the AV2F dataset, higher accuracies are achieved for classifying speed. Overall, the highest scores are reached for classifying agent types, as on the AV2F dataset.



(a) Speed **(b)** Acceleration **(c)** Direction **(d)** Agent type

Figure 4.4: Linear probing accuracies of RedMotion for hidden states of module 0, module 1 and module 2 on the validation split of the Waymo dataset. Adapted from Tas and Wagner (2025).

4.2 Fitting interpretable control vectors

The previous experiments show that our features are linearly separable. While this indicates that the feature states have distinct representations, it does not allow us to modify these features to demonstrate their impact on motion forecasts. Therefore, this section focuses on fitting control vectors that enable us to modify the state of our categorical features.

We use our interpretable and categorical features to form pairs of opposing feature states, i.e. low and high speed. For each pair, we build a dataset and extract the corresponding hidden states of the last past motion token (temporal index $t = -1$). We choose this index as it is closest to the start of the forecast. Next, we compute the element-wise difference $\boldsymbol{d}$ between the hidden states of samples with opposing feature states (**pos**itive and **neg**ative examples)

$$\boldsymbol{d} = \boldsymbol{H}^{\text{pos}}_{-1,:} - \boldsymbol{H}^{\text{neg}}_{-1,:} \in \mathbb{R}^d, \tag{4.1}$$

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{d}_1^\top \\ \boldsymbol{d}_2^\top \\ \vdots \\ \boldsymbol{d}_N^\top \end{bmatrix} \in \mathbb{R}^{N \times d}, \tag{4.2}$$

where N is the number of pairs and d is the hidden dimension. Finally, we follow Zou et al. (2023) and apply principal component analysis (PCA) with a single component as a pooling method. This reduces the computed differences to a single scalar per hidden dimension to generate control vectors:

$$\boldsymbol{V}_{-1,:} = \text{PCA}(\boldsymbol{D}) = \arg \max_{\|\boldsymbol{V}_{-1,:}\|=1} \boldsymbol{V}_{-1,:}^\top \text{cov}(\boldsymbol{D}) \boldsymbol{V}_{-1,:} \in \mathbb{R}^d. \tag{4.3}$$

Therefore, $\boldsymbol{V}_{-1,:}$ is the eigenvector of the largest eigenvalue of $\text{cov}(\boldsymbol{D})$ and describes the most salient direction between positive and negative hidden states.

We repeat the previous steps for each analyzed feature to generate control vectors.

4.3 Modifying hidden states at inference

4.3.1 Experimental setup

We build pairs of opposing features for the AV2F and the Waymo dataset. Afterwards, we fit sets of control vectors as described in Section 4.2. At inference, we add the control vectors generated for the last temporal index ($t = -1$) to all embeddings ($t \in \{0, \dots, 49\}$ for AV2F, $t \in \{0, \dots, 10\}$ for Waymo) with

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{V}_{-1,:}^{\top} \\ \boldsymbol{V}_{-1,:}^{\top} \\ \vdots \\ \boldsymbol{V}_{-1,:}^{\top} \end{bmatrix} \in \mathbb{R}^{T \times d}, \tag{4.4}$$

where T is the number of past time steps and d is the hidden dimension.

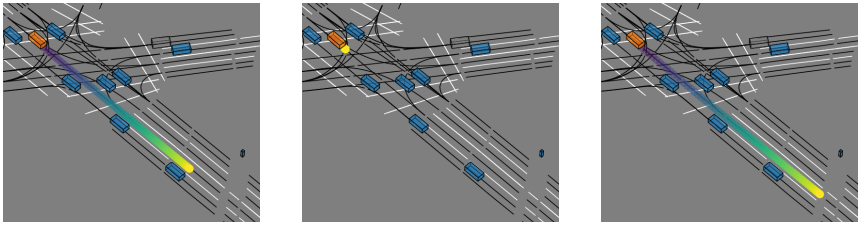
Furthermore, we use a temperature parameter τ to scale our control vectors and the corresponding control signal

$$\boldsymbol{H}_{\text{modified}} = \boldsymbol{H} + \tau \boldsymbol{V}. \tag{4.5}$$

4.3.2 Qualitative results

Figure 4.5 shows a qualitative example from the AV2F dataset, where we modify hidden states using our control vector for acceleration scaled with different temperatures τ. Subfigure 4.5a shows the default (i.e., non-controlled) top-1 (i.e., most likely) motion forecast. In subfigures 4.5b and 4.5c, we apply our acceleration control vector with $\tau = -20$ and $\tau = 100$ to enforce a strong deceleration and a moderate acceleration, respectively.

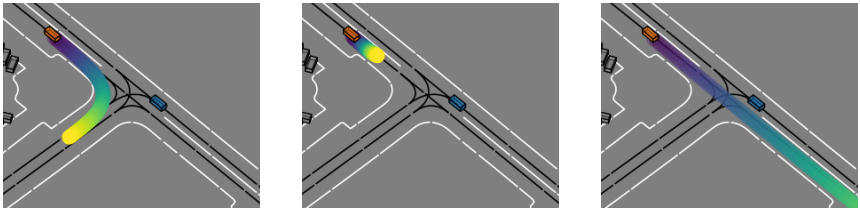
Figure 4.6 shows a qualitative example from the Waymo dataset. Subfigure 4.6a shows the default motion forecast. In subfigures 4.6b and 4.6c, we apply our



(a) Default motion forecast **(b)** Acceleration control $\tau = -20$ **(c)** Acceleration control $\tau = 100$

Figure 4.5: Modifying hidden states to control a vehicle at an intersection. We add our acceleration control to enforce a strong deceleration and a moderate acceleration. The controlled agent is highlighted in orange, dynamic agents are blue, and static agents are grey. Lanes are black lines and road markings are white lines. Adapted from Tas and Wagner (2025).

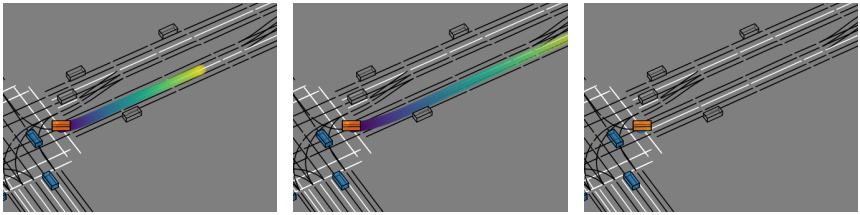
speed control vector to decrease and increase the driven speed of a vehicle. Both modifications affect the future speed in a similar manner, while increasing the speed also changes the route to fit the given environment context (i.e., lanes).



(a) Default motion forecast **(b)** Speed control $\tau = -32$ **(c)** Speed control $\tau = 100$

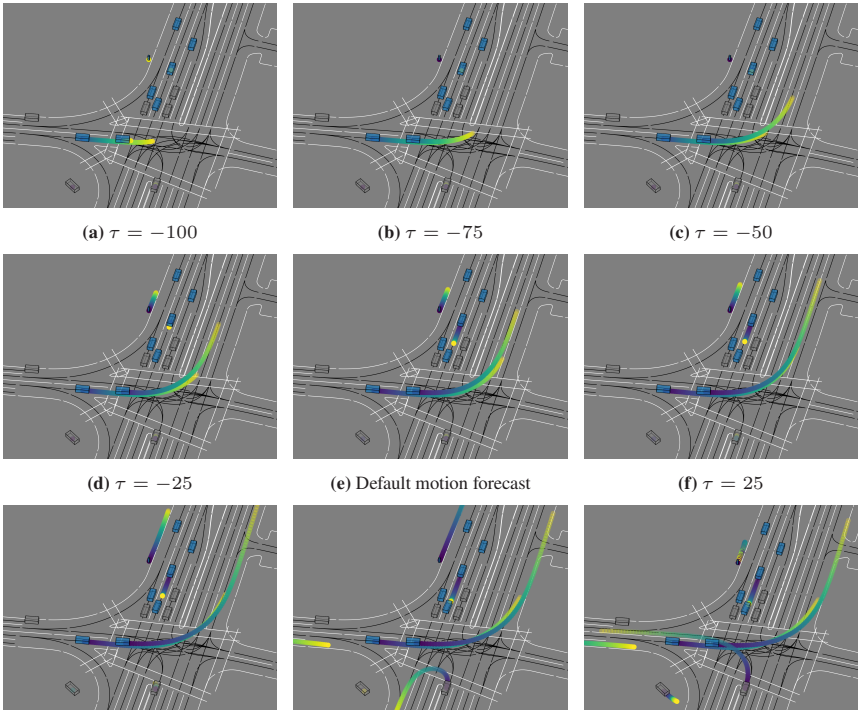
Figure 4.6: Modifying hidden states to control a vehicle before a predicted right turn. Increasing the speed also changes the route to fit the given context (i.e., lanes). Adapted from Tas and Wagner (2025).

Figure 4.7 shows a qualitative example for our direction control vector from the Argoverse 2 Forecasting dataset. The left control leads to accelerated future motion, which is consistent with the common driving style of slowing down in front of a curve and accelerating again when exiting the curve. A strong right control makes the focal agent stationary. We hypothesize that it cancels out the actually driven left turn, resulting in a virtually stationary past.



(a) Default motion forecast **(b)** Left control $\tau = 10$ **(c)** Right control $\tau = 100$

Figure 4.7: Modifying hidden states to control a left turning vehicle. We apply our left-direction control vector and right-direction control vector. Adapted from Tas and Wagner (2025).



(a) $\tau = -100$ **(b)** $\tau = -75$ **(c)** $\tau = -50$

(d) $\tau = -25$ **(e)** Default motion forecast **(f)** $\tau = 25$

(g) $\tau = 50$ **(h)** $\tau = 75$ **(i)** $\tau = 100$

Figure 4.8: Speed control vector applied to multiple agents. Linearly scaling the control temperature parameter τ leads to rather linear changes in future speeds as well. Adapted from Tas and Wagner (2025).

Figure 4.8 shows an example from the Waymo dataset, where we apply our speed control vector to multiple agents in a scenario. Linearly scaling the temperature parameter leads to rather linear changes in future speeds as well. Furthermore, our control vectors can be applied pedestrians as well, see top middle in Figure 4.8. For $\tau = 100$, the forecasted pedestrian trajectory is unrealistic. This is likely because the model has learned an upper bound for how fast a pedestrian can move, and this bound is much lower than that for vehicles.

4.4 Improving control vectors via sparse autoencoding

In this section, we use sparse autoencoders (SAEs) to improve our control vectors. SAEs learn to encode and decode representations using sparse intermediate representations. Due to the enforced sparsity of intermediate representations, features are encoded in fewer, more distinct dimensions (see Section 2.2.3). This aligns with our objective of fitting control vectors to well-separated feature representations. Therefore, we propose fitting control vectors to the sparse intermediate representations learned by SAEs (see Figure 4.9).

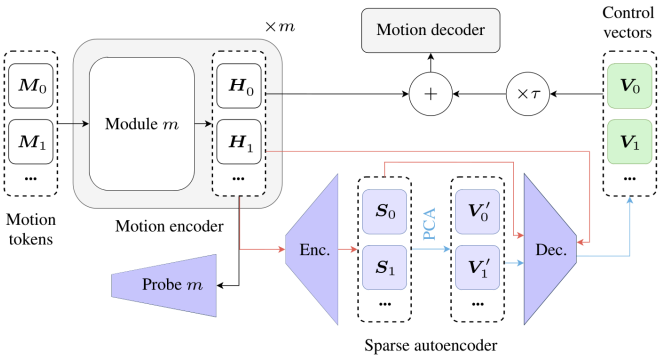


Figure 4.9: Words in Motion with sparse autoencoding. The training of the sparse autoencoder is shown with red arrows (→) and the fitting of control vectors with blue arrows (→).

In detail, we train an SAE and use its encoder to generate sparse intermediate representations of hidden states with opposing feature states ($\boldsymbol{H}^{\text{pos}}_{t,:}$ vs. $\boldsymbol{H}^{\text{neg}}_{t,:}$). Formally, we compute

$$\boldsymbol{S}^{\text{pos}}_{t,:} = \text{ReLU}\Big(\boldsymbol{W}_{\text{enc}}\big(\boldsymbol{H}^{\text{pos}}_{t,:} - \boldsymbol{b}_{\text{dec}}\big) + \boldsymbol{b}_{\text{enc}}\Big), \tag{4.6}$$

where $\boldsymbol{W}$ and $\boldsymbol{b}$ denote weights and biases of the SAE. Similarly, we compute $\boldsymbol{S}^{\text{neg}}_{i,:}$ and obtain the intermediate control vectors as

$$\boldsymbol{V}'_{t,:} = \text{PCA}\big(\boldsymbol{S}^{\text{pos}}_{t,:} - \boldsymbol{S}^{\text{neg}}_{t,:}\big). \tag{4.7}$$

Leveraging the Johnson-Lindenstrauss Lemma[2], we use the SAE decoder to project the intermediate control vectors back to the hidden dimension of the motion encoder

$$\boldsymbol{V}_{t,:} = \boldsymbol{W}_{\text{dec}}\boldsymbol{V}'_{t,:} + \boldsymbol{b}_{\text{dec}}. \tag{4.8}$$

This enables using sparse autoencoders of arbitrary sparse intermediate dimensions for generating control vectors of fixed dimension. Consequently, we do not use the SAE at inference as the resulting control vectors match the hidden dimension of the analyzed models and can be directly applied (as in our baseline method in Section 4.2).

4.4.1 Experimental setup

We measure the quality of a control vector using linearity measures. Specifically, we measure the linearity of relative speed changes in forecasts when scaling speed control vectors (see Figure 4.8). This evaluation is motivated by the fact that linear effects enable an intuitive interface for controlling models during inference. We use the Pearson correlation coefficient, the coefficient of determination (R^2), and the straightness index (S-idx) (Benhamou 2004) as linearity measures. Given the

[2] Johnson and Lindenstrauss (1984) state that a set of points in high-dimensional space can be projected into a lower-dimensional space while approximately preserving the pairwise distances between points.

large range of scenarios in the Waymo dataset, we focus on relative speed changes within a range of $\pm 50\%$ (see Appendix A.4). All experiments are performed on the Waymo dataset and using our RedMotion model as motion forecasting model.

4.4.2 Results

We compute linearity measures for control vectors optimized using regular SAEs (Bricken et al. 2023) with varying sparse intermediate dimensions. We achieve the highest scores using the SAE with a dimension of 128 (see Table 4.1). Therefore, we use this dimension in the rest of our evaluations.

In the following, we evaluate autoencoders with different activation functions and layer types. Following Rajamanoharan et al. (2024), we use JumpReLU with a threshold $\theta = 0.001$ and regular ReLU activation functions. Moreover, we evaluate regular SAEs with fully-connected layers, with MLPMixer (Tolstikhin et al. 2021) layers (Sparse MLPMixer), and with convolutional layers (ConvSAE). For Sparse MLPMixer and ConvSAE, we use large patch and kernel sizes to approximate the global receptive fields of fully-connected hidden units in regular SAEs. Furthermore, we evaluate a consistent Koopman autoencoder (KoopmanAE) (Azencot et al. 2020) to include a method that models temporal dynamics between embeddings (see Appendix A.5).

Autoencoder	Pearson	R^2	S-idx
SAE-512	0.990	0.974	0.984
SAE-256	0.990	0.974	<u>0.985</u>
SAE-128	**0.993**	**0.984**	**0.988**
SAE-64	<u>0.991</u>	<u>0.976</u>	<u>0.985</u>
SAE-32	0.990	0.959	<u>0.985</u>
SAE-16	0.982	0.770	0.958

Table 4.1: Scaling sparse autoencoders. Adapted from Tas and Wagner (2025).

Table 4.2 presents linearity measures for different control vectors derived from both plain PCA pooling and SAE methods. Overall, the regular SAEs achieve the highest Pearson and R^2 scores. JumpReLU activation functions improve the R^2 scores marginally compared to ReLU activation functions. The SAE version of Cunningham et al. (2023) does not improve the linearity scores. We hypothesize

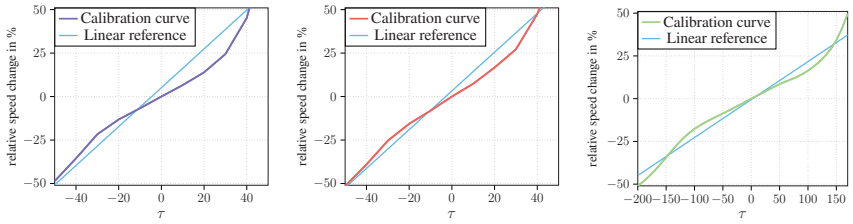
that this is due to reduced decoding flexibility since they transpose the encoder weights instead of learning the decoder weights (i.e., $\boldsymbol{W}_{\text{dec}} = \boldsymbol{W}_{\text{enc}}^{\top}$).

The ConvSAE with a kernel size $k = 64$ and the KoopmanAE achieve the highest straightness index, yet the lowest R^2 scores. As shown in Figure 4.10 and Figure A.1 in the appendix, the range of temperatures τ is much higher for this ConvSAE and significantly lower for the KoopmanAE than for e.g. the regular SAE. This lowers the R^2 score but does not affect the straightness index.

For the ConvSAE, we hypothesize that this is due to strong activation shrinkage (Rajamanoharan et al. 2024). Therefore, the JumpReLU configuration of this SAE-type leads to a significantly smaller τ range (see Figure 4.11), which in turn leads to higher R^2 scores (see Table 4.2).

For the KoopmanAE, the opposite is likely, since activation shrinkage is caused by sparsity terms, which are not included in the loss function of Azencot et al. (2020).

Notably, activation steering with our SAE-based control vector has an almost 1-to-1 ratio between τ and relative speed changes (i.e., $\tau = -50$ corresponds to roughly -50%). This improves R^2 scores and enables an intuitive interface. Furthermore, improved controllability with SAEs indicates that sparse intermediate representations capture more distinct features.

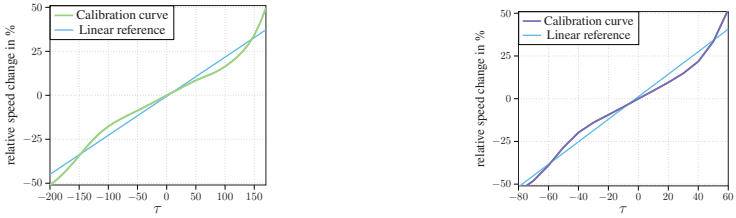


(a) Plain PCA **(b)** SAE **(c)** ConvSAE $k = 64$

Figure 4.10: Calibration curves of plain PCA-based speed control vectors and control vectors optimized using SAEs for relative speed changes in forecasts of $\pm 50\%$. Adapted from Tas and Wagner (2025).

Autoencoder	Activation function	Pooling	Patch/kernel size	Pearson	R^2	S-idx
–	–	PCA	–	0.988	0.969	0.981
SAE (Bricken et al. 2023)	ReLU	PCA	–	**0.993**	0.984	0.988
SAE (Rajamanoharan et al. 2024)	JumpReLU	PCA	–	**0.993**	**0.986**	0.988
SAE (Cunningham et al. 2023)	ReLU	PCA	–	0.987	0.971	0.980
Sparse MLPMixer	ReLU	PCA	64	0.992	0.980	0.986
Sparse MLPMixer	JumpReLU	PCA	64	0.992	0.981	0.986
Sparse MLPMixer	ReLU	PCA	32	0.990	0.978	0.985
Sparse MLPMixer	JumpReLU	PCA	32	0.991	0.980	0.986
ConvSAE	ReLU	PCA	64	0.986	0.383	0.991
ConvSAE	JumpReLU	PCA	64	0.987	0.861	0.978
ConvSAE	ReLU	PCA	32	0.988	0.622	0.986
ConvSAE	JumpReLU	PCA	32	0.989	0.623	0.986
KoopmanAE (Azencot et al. 2020)	tanh	PCA	–	0.991	−0.057	**1.000**

Table 4.2: Comparing control vectors optimized with different autoencoders. Linearity measures for optimized control vectors: Pearson correlation coefficient, coefficient of determination (R^2), and straightness index (S-idx). Adapted from Tas and Wagner (2025).



(a) ConvSAE $k = 64$ **(b)** ConvSAE $k = 64$ JumpReLU

Figure 4.11: JumpReLU compensates activation shrinkage as reflected in a smaller range of τ values for the same range of relative speed changes. Adapted from Tas and Wagner (2025).

Furthermore, we perform an ablation study analyzing our method's sensitivity to hidden states from different modules (see Table 4.3) and to varying speed thresholds (see Table 4.4). Our method performs best with a sparse intermediate dimension of 128 and hidden states from module $m = 2$; and is more sensitive to low than to high speed thresholds.

Autoencoder	Module m	Pearson	R^2	S-idx
SAE-128	2	**0.993**	**0.984**	**0.988**
SAE-128	1	0.992	0.980	0.987
SAE-128	0	0.959	0.654	0.933

Table 4.3: Generating control vectors for hidden states of different modules. Control vectors for speed generated in earlier modules achieve lower linearity scores. Linearity measures for controlling: Pearson correlation coefficient, coefficient of determination (R^2), and straightness index. Adapted from Tas and Wagner (2025).

Autoencoder	Low speed	High speed	Pearson	R^2	S-idx
SAE-128	$< 25\,\mathrm{km/h}$	$> 50\,\mathrm{km/h}$	0.993	0.984	0.988
SAE-128	$< 25\,\mathrm{km/h}$	25 to $50\,\mathrm{km/h}$	0.994	0.987	0.989
SAE-128	25 to $50\,\mathrm{km/h}$	$> 50\,\mathrm{km/h}$	0.355	−0.734	0.533

Table 4.4: Generating speed control vectors with different thresholds for low and high speed. Decreasing the threshold for high speed marginally improves linearity scores, while increasing the threshold for low speed significantly worsens the linearity scores. Adapted from Tas and Wagner (2025).

4.5 Connecting neural collapse and controllability

We train a RedMotion model on the AV2F dataset using the same trajectory lengths as in the Waymo dataset (1.1 s past and 8 s future). Table 4.5 shows the probing accuracy and linearity measures of a speed control vector for this model (see Figure 4.12 for the calibration curve). Additionally, we compute the neural collapse metric class-distance normalized variance (CDNV) (Galanti et al. 2021). CDNV quantifies the degree to which features form class-wise clusters by measuring the variance within feature clusters of each class c relative to the distances between class means μ_c. CDNV provides a robust alternative to methods that compare between- and within-cluster variance (Papyan et al. 2020).

$$\mathrm{CDNV}(c, c') = \frac{\sigma_c^2 + \sigma_{c'}^2}{2\|\mu_c - \mu_c'\|_2^2}, \quad \forall c \neq c' \tag{4.9}$$

Compared with a model trained on the Waymo dataset, the AV2F model achieves both worse neural collapse metrics (probing accuracy and CDNV) and significantly lower linearity measures. These results support our argument that latent space regularities with separable features, like neural collapse, are necessary to fit precise control vectors.

Dataset	Probing accuracy↑	CDNV↓	Pearson↑	R^2↑	S-idx↑
AV2F	0.753	2.46	0.877	0.275	0.891
Waymo	0.945	0.95	0.988	0.969	0.981

Table 4.5: Higher probing accuracy enables higher linearity measures. We train RedMotion models on the Waymo and AV2F datasets using the same trajectory lengths. We report the probing accuracies and CDNV for speed feature states and the linearity measures for the corresponding PCA-based control vectors.

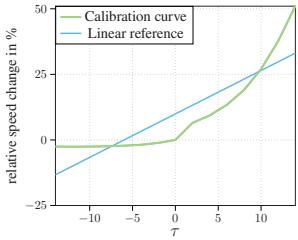
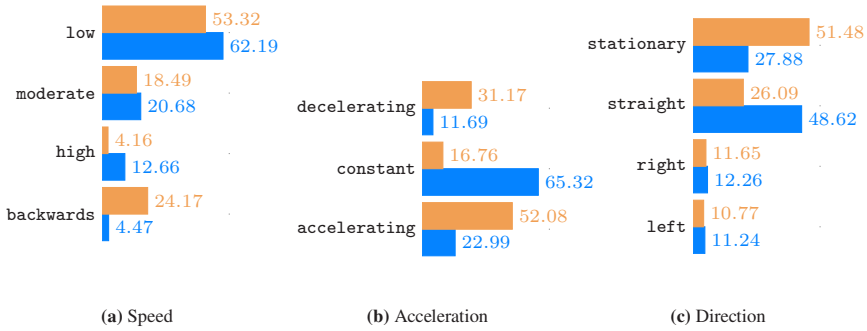


Figure 4.12: Calibration curve for a plain PCA-based speed control vector for the AV2F dataset. In contrast to the control vectors for the Waymo dataset, this control vector cannot reduce the speed by more than 3%. Adapted from Tas and Wagner (2025).

Moreover, we used the low and moderate speed states to fit this control vector, as the low and high speed states did not yield good results. We hypothesize that this is due to the different distributions of the datasets shown in Figure 4.13.

Both datasets predominantly capture low-speed scenarios, with 62% of Waymo instances and 53% of AV2F instances falling into this category. Furthermore, a notable difference lies in the proportion of stationary vehicles, with AV2F exhibiting a significantly higher percentage (51%) compared to Waymo (28%). The Waymo dataset predominantly features vehicles with constant acceleration



(a) Speed **(b)** Acceleration **(c)** Direction

Figure 4.13: Distributions of our categorical motion features for the AV2F and the Waymo dataset. All numbers are percentages. Adapted from Tas and Wagner (2025).

(65%) and traveling straight (49%), while the AV2F dataset has a higher proportion of accelerating instances (52%). Additionally, AV2F has a much larger proportion of instances involving backward motion (24%) compared to Waymo (4%). This disparity in motion characteristics highlights that the two datasets capture different driving environments and scenarios, with Waymo focusing on urban and suburban driving, while AV2F contains only urban traffic scenarios (see Section 2.1.5).

4.6 Fuzz testing control vector temperatures

In this experiment, we perform fuzz testing of control vector temperatures τ for our speed control vectors. Specifically, we test a greater range of τ values $[-200, 200]$ to evaluate whether our modifications can lead to unrealistic trajectory forecasts. Unlike our previous experiments, we perform fuzz testing for each agent type individually.

We measure how realistic trajectory forecasts are by measuring jerk and tortuosity (see Section 2.1.4). As reference, we initially measure the average jerk and tortuosity of ground truth trajectories on the Waymo dataset. Table 4.6 shows the results. Pedestrians experience the least jerk and the most tortuosity. Therefore, they change acceleration the least, but direction the most.

Agent type	Jerk	Tortuosity
Vehicle	$1.075\,\mathrm{m/s^3}$	1.284
Pedestrian	$0.808\,\mathrm{m/s^3}$	1.869
Cyclist	$1.490\,\mathrm{m/s^3}$	1.217

Table 4.6: Average jerk and tortuosity of ground truth trajectories of the Waymo dataset.

Afterwards, we compute the average metrics for trajectory forecast when modifying the speed feature with different control temperatures τ. We use the SAE-128 optimized control vector for our RedMotion model. The plots for the corresponding plain PCA-based control vector are very similar and shown in Appendix A.6.

The unmodified forecasts for $\tau = 0$ have lower jerk and tortuosity values than the ground truth trajectories, indicating that our model forecasts smoother yet more basic maneuvers.

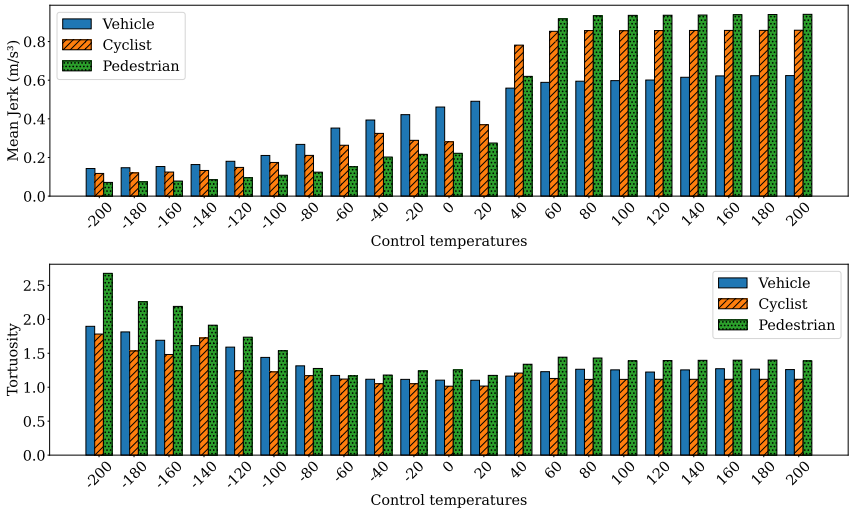


Figure 4.14: Fuzz testing control temperatures τ for SAE-128 optimized speed control vectors. All metrics are for motion forecasts of our RedMotion model on the Waymo dataset.

When we increase the control temperature to about 60, jerk and tortuosity reach similar levels as for the ground truth. When further increasing τ to up to 200,

both jerk and tortuosity seem to plateau. Decreasing τ to -200, yields unrealistic trajectories with high tortuosity values above 2 for pedestrians. For comparison, a u-turn modeled as a semicircle has the highest tortuosity value of standard maneuvers, at approx. 1.6. Therefore, trajectories with higher tortuosity values are very winding and likely unrealistic.

Overall, agents of type pedestrian seem to be most sensitive to our control vector since both jerk and tortuosity vary the most. At inference, we suggest computing jerk and tortuosity for all trajectories as safety validation when modifying learned feature representations.

4.7 Compensating for domain shifts with control vectors

An application of control vectors that goes beyond mechanistic interpretability is compensating for domain shifts. Domain shifts between training and test data significantly degrade the performance of many data-driven methods. Zero-shot generalization methods compensate for such domain shifts without further training (e.g., Xian et al. 2017, Mistretta et al. 2024).

In motion forecasting, common domain shifts are more or less aggressive driving styles that result in higher or lower speeds, respectively. We simulate this domain shift by reducing the future speed in the Waymo validation split by approximately 50%. Specifically, we take the first half of future waypoints and linearly upsample this sequence to the original length.

Table 4.7 shows the results of a RedMotion model trained on the regular training split on this validation split with domain shift. We provide an overview of the used motion forecasting metrics in Section 2.1.4. Without the use of our control vectors, high distance-based errors, miss, and overlap rates are obtained. Using the calibration curve in Figure 4.10b, we compensate for this domain shift by applying our SAE-128 control vector with a temperature $\tau = -50$. This significantly reduces the distance-based errors, the overlap, and the miss rates

without further training. In addition, we show the results of applying our control vector with a temperature of $\tau = -30$ and $\tau = -70$, which improves all scores over the baseline (first row) as well.

Control vector	Temperature τ	minADE↓	Brier minADE↓	minFDE↓	Brier minFDE↓	Overlap rate↓	Miss rate↓
None		3.271	6.547	4.617	8.933	0.220	0.580
SAE-128	-30	<u>1.685</u>	4.838	2.870	8.429	<u>0.179</u>	**0.224**
SAE-128	-50	**1.174**	**2.759**	**1.798**	**4.329**	**0.174**	<u>0.236</u>
SAE-128	-70	1.808	<u>3.576</u>	<u>2.035</u>	<u>3.676</u>	0.189	0.302

Table 4.7: Zero-shot generalization to a Waymo dataset version with reduced future speeds. Best scores are **bold**, second best are <u>underlined</u>. Adapted from Tas and Wagner (2025).

5 Instructable retrocausal motion forecasting

Control vectors provide control over high-level features, such as speed, acceleration, or agent type (see Chapter 4). In this chapter, we propose a motion forecasting model, which can be controlled using more fine-grained instructions.

Specifically, we propose a multi-task learning method for motion forecasting that includes a retrocausal flow of information. The corresponding tasks are to forecast (1) marginal trajectory distributions for all modeled agents and (2) joint trajectory distributions for pairs of interacting agents. Our model generates the joint trajectory distributions by re-encoding marginal trajectory distributions and subsequent pairwise modeling. This incorporates a retrocausal flow of information directed from later points in marginal trajectories to earlier points in joint trajectories.

Notably, we find that this retrocausal information flow can be intercepted and modified to issue goal-based and directional instructions. This shows that the learned retrocausal connections are functionally important and enables mechanistic interpretations. Furthermore, the ability to follow instructions emerges without explicit supervision since we train our model for regular motion forecasting without using instructions.

5.1 Method

We probabilistically model positional uncertainty in future trajectories. Specifically, our model uses mixture distributions to decompose motion forecasts in the following three ways.

5.1.1 Decomposing exponential power distributions

For each trajectory point (i.e., future position), we model the positional uncertainty using the probability density function[1] of an exponential power distribution[2]. The exponential power distribution is a parametric family of probability distributions, which includes all continuous uniform distributions (e.g., normal or Laplace distributions). In addition to a location μ and a scale parameter σ, it is characterized by a shape parameter β.

We follow common practice (Nayakanti et al. 2023, Zhou et al. 2023a,b) and model the x and y coordinates of future positions as uncorrelated random variables. Therefore, the density of the corresponding bivariate exponential power distribution is

$$\mathcal{D}'(x, y; \boldsymbol{\mu}, \boldsymbol{\sigma}, \beta) = \frac{\beta^2}{4\sigma_x \sigma_y [\Gamma(1/\beta)]^2} \exp\left(- \left| \frac{x - \mu_x}{\sigma_x} \right|^\beta - \left| \frac{y - \mu_y}{\sigma_y} \right|^\beta \right), \tag{5.1}$$

where $\Gamma(\cdot)$ is the gamma function.

Figure 5.1 shows how β affects the shape of the density of this bivariate distribution. In all plots, we set $\mu_x = \mu_y = 0$ and $\sigma_x = \sigma_y = 1$. Starting from a standard normal distributions with $\beta = 2$, decreasing β narrows the peak and increases the weight of the tails. Increasing β decreases the weight of distribution tails and flattens the peak.

[1] Abbreviated as density.
[2] Also known as symmetric generalized normal distributions.

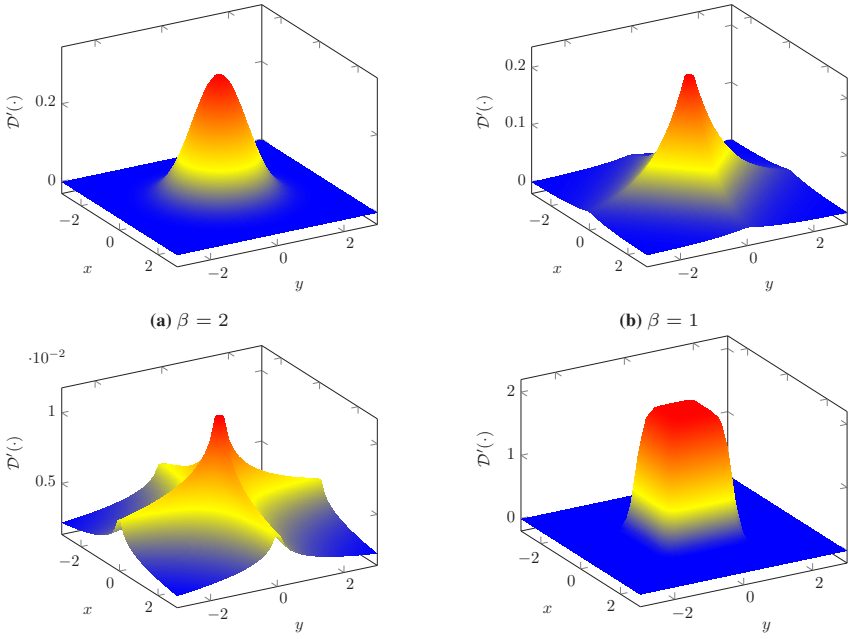


Figure 5.1: Bivariate exponential power distributions with $\beta = 2$ (normal), $\beta = 1$ (Laplace), $\beta = 0.25$ (heavy-tailed), and $\beta = 8$ (platykurtic).

Heavy-tailed and platykurtic densities (characterized by flatter peaks) create substantial uncertainties near forecasted positions, which is undesirable in subsequent motion planning (cf. Taş et al. 2023). Therefore, we clamp the shape parameter β to the range of $[1.0, 2.0]$. We implement this constraint by approximating exponential power distributions using mixture distributions of bivariate normal and bivariate Laplace distributions. The corresponding mixture density is

$$\mathcal{D}(x, y; w, \boldsymbol{\phi}) = w \cdot \text{normal}(x, y; \boldsymbol{\phi}) + (1 - w) \cdot \text{Laplace}(x, y; \boldsymbol{\phi}), \qquad (5.2)$$

with learned weights $0 \leq w \leq 1$ and density parameters $\boldsymbol{\phi} = (\mu_x, \mu_y, \sigma_x, \sigma_y)$. For brevity, we include w in the tuple of density parameters $\boldsymbol{\phi}$ in the following formulas.

5.1.2 Decomposing marginal trajectory distributions

We train a distinct decoder to perform marginal motion forecasting (i.e., per-agent). We follow common practice (Chai et al. 2020, Zhang et al. 2023, Nayakanti et al. 2023) and predict the density of a mixture distribution at each future time step t and fix mixture weights over time. Per-agent mixture components describe future positions of the same agent and at the same time, but from different trajectories. Formally, we follow Bishop (1994) and express this as

$$\mathcal{P}_t^{\text{marginal}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) = \sum_{k=1}^{K} m_k(\boldsymbol{x}; \boldsymbol{\theta}) \cdot \mathcal{D}\big(\boldsymbol{y} \mid \boldsymbol{\phi}_{t,k}(\boldsymbol{x}; \boldsymbol{\theta})\big), \tag{5.3}$$

where $\boldsymbol{x}$ is the input (cf. Section 5.1.5), $\boldsymbol{y}$ the target vector (i.e., ground truth trajectory), $\boldsymbol{\theta}$ are the parameters of our model, $t \in \{1, ..., T\}$ are future time steps, K is the number of trajectories, k indexes the corresponding mixture components, and $\boldsymbol{m}$ are mixture weights. Unlike mixture weights, density parameters $\boldsymbol{\phi}$ are variable across mixture components and time steps.

5.1.3 Decomposing joint trajectory distributions

We re-encode marginal distributions (see Section 5.1.2) to perform pairwise modeling. To exchange information between agents, we transform all trajectories to the local reference frame of agent 1 and use the scene context embeddings of agent 1 for agent 2 as well (see Section 5.1.5). Afterwards, we exchange information via attention mechanisms and decode joint trajectory distributions using multi-agent mixture components (see Figure 5.2). Per time step t, each

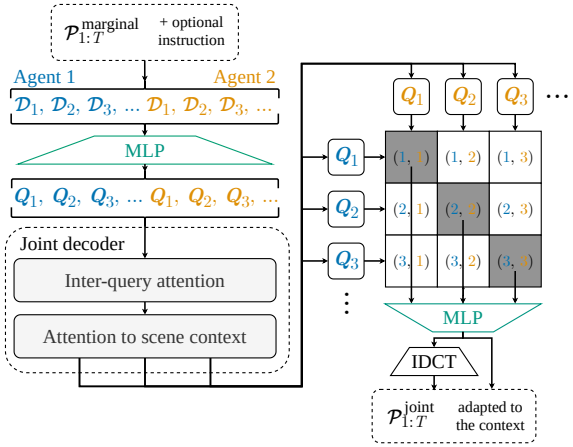


Figure 5.2: From marginal to joint trajectories. We use an MLP to generate query matrices $\boldsymbol{Q}$ from marginal trajectories and exchange information between queries and scene context with attention mechanisms. Afterwards, we decode joint trajectories $\mathcal{P}^{\text{joint}}_{1:T}$ from pairs of queries at the same index. This compresses information from all K^2 possible combinations into K query pairs. Adapted from Wagner et al. (2025).

mixture component is a mixture density itself, representing one future position of each agent

$$\mathcal{P}^{\text{joint}}_t(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) = \sum_{k=1}^{K} c_k \sum_{a=1}^{A} M_{k,a}(\boldsymbol{x}; \boldsymbol{\theta}) \cdot \mathcal{D}\big(\boldsymbol{y} \mid \boldsymbol{\phi}_{t,k,a}(\boldsymbol{x}; \boldsymbol{\theta})\big), \tag{5.4}$$

where A is the number of agents and $\boldsymbol{M}$ is a matrix of per-agent mixture weights. We compute multi-agent mixture weights with $\boldsymbol{c} = \text{softmax}\big(\sum_{a=1}^{A} \boldsymbol{M}_{1:K,a}/\tau\big)$ and a tunable temperature parameter τ. Thus, τ controls the entropy of the distribution, with low values skewing the distribution toward its peaks.

As shown in Figure 5.2, this approximates the joint distribution of all combinations of per-agent mixture components by focusing on the diagonal query pairs in matrix form. Specifically, we decode joint trajectory distributions using only the on-diagonal query pairs. Off-diagonal queries can update on-diagonal queries

through attention mechanisms (see inter-query attention in Figure 5.2), which compresses information from all K^2 possible combinations into K query pairs.

For both marginal and joint motion forecasts, we follow related methods (Chai et al. 2020, Shi et al. 2022, Zhou et al. 2023a) and reuse the out-most mixture weights (m_k and c_k in Equation (5.3), Equation (5.4)) as predicted probabilities.

5.1.4 Compressing location parameters of probability densities

Regressing trajectories at a high frequency of 10 Hz (e.g., Zhou et al. 2023a, Zhang et al. 2023) allows models to predict sudden changes between successive time steps, which are physically impossible yet close to the ground truth (see Section 2.1.8). Therefore, distance-based loss functions do not penalize such errors as much.

To compensate for this issue, we use a compressed probabilistic representation of trajectories that excludes high-frequency components. Specifically, we append the inverse discrete cosine transform (IDCT) to our model to internally represent density location parameters as a sum of cosine functions (see Figure 5.2). We hypothesize that this is a natural choice for transformer models given the common use of cosine-like positional encodings (e.g., sinosoidal (Vaswani et al. 2017) or rotary positional encodings (Su et al. 2024)). To compress, we limit the frequencies in the IDCT to the lower end. This compression method is data-independent and thus invariant to dataset or setup-specific noise (e.g., produced by errors of perception models).

5.1.5 Scene encoder

We follow Gao et al. (2020) and represent multimodal inputs (i.e., past trajectories, lane data, and traffic light states) as polyline vectors. We sample temporal features (past positions and traffic light states) with a frequency of 10 Hz and static spatial

features (lane markings and road borders) with a resolution of 0.5 meters. We generate embeddings for each modality with 3-layer MLPs, add sinusoidal positional encodings, and process the embeddings with transformer encoder modules (Vaswani et al. 2017). Following Nayakanti et al. (2023), we initially process local agent-centric views within scenes (centered around each modeled agent) and compress them using cross-attention. We then change the batch dimension from the agent index to the scene index, and concatenate learned embeddings of Cartesian transformation matrices from agent-centric views into a global reference frame (cf. Jiang et al. (2023)). Finally, we add global sinosoidal positional encodings and generate global scene context representations with further self-attention mechanisms. Our two decoders (Section 5.1.2 and Section 5.1.3) decode trajectory distributions from this scene context.

5.1.6 Loss function

We train our model using maximum likelihood estimation with a multi-task loss that covers the objectives described in Section 5.1.2 and Section 5.1.3. Specifically, we minimize the negative log-likelihood for forecasting the ground truth trajectories.

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} -w_n^{\text{type}} \ln \left(\mathcal{P}_{t,k=\hat{k}}^{\text{joint}} \right) - \lambda_{\text{marginal}} \ln \left(\mathcal{P}_{t,k=\hat{k}}^{\text{marginal}} \right), \tag{5.5}$$

where N is number of samples in a batch, w_n^{type} is a vector of configurable loss weights per agent type[3], $\lambda_{\text{marginal}}$ is a tunable weighting factor. We optimize this objective with multiple trajectories per agent by backpropagating only the error for the trajectories that are closest to the ground truth trajectories[4]. We measure the

[3] Inspired by the evaluation protocol of Ettinger et al. (2021), we use the weight of the least common agent type involved in scene. Overall, cyclists are least common, pedestrians are more common, and vehicles are most common. By default, we set the weight for all agent types to 1.

[4] Also referred to as winner-takes-all (Shi et al. 2022) or loss with hard assignment (Zhang et al. 2023).

distance to the ground truth using the L_2-norm. For marginal forecasts $\mathcal{P}_{1:T}^{\text{marginal}}$, we select the best trajectory index $\hat{k}$ per agent. For joint forecasts $\mathcal{P}_{1:T}^{\text{joint}}$, we select the best set of trajectories at the same index $\hat{k}$ for agent pairs (see Figure 5.2).

5.2 Experiments

In this section, we evaluate the motion forecasting performance of our RetroMotion method using the Waymo and AV2F datasets (see section 2.1.5). Afterwards, we show that regular training of motion forecasting leads to the ability to follow goal-based instructions and to adapt basic directional instructions to the scene context. Furthermore, we analyze learned density representations using forecasting metrics, mixture weights, and metrics for neural regression collapse (Andriopoulos et al. 2024).

5.2.1 Interactive motion forecasting

5.2.1.1 Model configuration

We configure our marginal decoder to forecast marginal trajectory distributions of 8 agents and our joint decoder to forecast joint trajectory distributions for 2 agents. Our scene encoder processes the 128 closest map polylines and up to 48 trajectories of surrounding agents per modeled agent. We append an IDCT transform to our model that reconstructs 80 location parameters (for x- and y-coordinates) from 16 predicted DCT coefficients.

We build a sparse mixture of experts model (SMoE) (see Section 2.2.5) of 3 variations of our model (see Section 5.2.1.3). All expert models are trained independently (see Section 5.2.1.2). At inference, we use a rule-based router that selects one expert model based on the agent type and perform non-maximum suppression on joint trajectory forecasts (see Section 5.2.1.4).

5.2.1.2 Training details

We sample 32 scenes from the Waymo dataset in a batch, with 8 focal (i.e., predicted) agents per scene. Specifically, we predict marginal trajectory distributions for all 8 agents and joint distributions for two interactive agents. We use Adam with weight decay (Loshchilov and Hutter 2019) as the optimizer and a step learning rate scheduler to halve the initial learning rate of 2×10^{-4} every 10 epochs. We train for 50 epochs using data distributed parallel (DDP) training on 4 Ada A6000 GPUs.

5.2.1.3 Expert model configuration

Our SMoE model contains 3 expert models adapted to the 3 evaluated agent types (vehicle, pedestrian, and cyclist). Inspired by the evaluation protocol of Ettinger et al. (2021), we use one expert for joint trajectory forecasts of only vehicles, one for forecasts that involve pedestrians and vehicles, and one for forecasts of cyclists and vehicles. The expert model for vehicles is trained to forecast distributions of 18 trajectories per agent. The pedestrian and cyclist expert models are trained to forecast 6 trajectories per agent. For the cyclist expert, we increase the loss weight of cyclist trajectories $10\times$ (see w^{type} in Equation (5.5)).

5.2.1.4 Post-processing

We follow Konev (2022) and reduce the predicted probability of redundant (i.e., similar) trajectories. We adapt this mechanism to joint trajectory distributions by suppressing probabilities only if a nearby trajectory belongs to a joint trajectory set with a higher accumulated probability (see c in Section 5.1.3). For the vehicle expert, we additionally perform top$k$-selection to first reduce the 18 predicted trajectories to 6 trajectories.

Then, we use beam search on the training split to determine the suppression thresholds per expert and agent class. Specifically, we expand[5] the most promising threshold of 10 initial thresholds per agent type. This includes tuning the softmax temperature τ used to compute the accumulated probabilities (see Section 5.1.3). For the BeTopNet-based cyclist expert, we use the default non-maximum suppression (NMS) of Shi et al. (2022). All distance thresholds and τ values for the results in Table 5.1 are in Appendix A.8.

5.2.1.5 Results

Table 5.1 presents motion forecasting metrics (see Section 2.1.4) for interactive forecasting on the Waymo Open Motion dataset. The QCNeXt model (Zhou et al. 2023b) performs strongly on the distance-based minADE and minFDE metrics. Averaged over all agent types, our RetroMotion model outperforms all other methods on the main metric mAP, indicating that our model predicts higher confidence scores for trajectories close to the ground truth.

Furthermore, we follow Ngiam et al. (2022) and evaluate the marginal predictions of our marginal decoder as joint predictions on the validation split (see marginal as joint configuration in Table 5.1). All forecasting metrics are significantly worse than for the joint predictions. This highlights the ability of our model to perform joint modeling by re-encoding and adapting the marginal predictions to the predictions of surrounding agents. Figure 5.3 shows qualitative results of our method on the validation split. We show joint and combined (marginal and joint, see bottom right) forecasts to highlight that our method is suitable for more complex scenarios with many agents.

Since the top-4 methods over all agent types in Table 5.1 are within one mAP, we further compare them in Table 5.2. Fully data-driven methods with learned anchor[6] initialization (QCNeXt and RetroMotion) tend to perform better on more

[5] Here expanding means that we fix this threshold and further optimize the threshold for other agent types and τ values.

[6] We refer to the initial vector representation of queries used to decode trajectories as anchors.

Split	Method (config)	Venue	mAP↑	minADE↓	minFDE↓	MR↓	OR↓
Test	Scene Transformer (joint) (Ngiam et al. 2022)	ICLR'22	0.1192	0.9774	2.1892	0.4942	0.2067
	GameFormer (joint) (Huang et al. 2023)	ICCV'23	0.1376	0.9161	1.9373	0.4531	0.2112
	JointMotion (HPTR) (Wagner et al. 2024a)	CoRL'24	0.1869	0.9129	2.0507	0.4763	0.2037
	MotionDiffuser (Jiang et al. 2023)	CVPR'23	0.1952	0.8642	1.9482	0.4300	0.2004
	JFP (Luo et al. 2023)	CoRL'23	0.2050	0.8817	1.9905	0.4233	0.1835
	MotionLM (Seff et al. 2023)	ICCV'23	0.2178	0.8911	2.0067	0.4115	0.1823
	MTR++ (Shi et al. 2024)	TPAMI'24	0.2326	0.8795	1.9509	0.4143	**0.1665**
	QCNeXt (Zhou et al. 2023b)		0.2352	**0.7750**	**1.6772**	**0.3838**	0.1946
	BeTopNet (Liu et al. 2024)	NeurIPS'24	0.2412	0.9744	2.2744	0.4355	0.1695
	RetroMotion (SMoE) (ours)		0.2422	0.9029	2.0245	0.4433	0.2007
	RetroMotion (SMoE hybrid) (ours)		**0.2519**	0.9256	2.0890	0.4347	0.1927
Val	MotionLM (single replica)	ICCV'23	0.1687	1.0345	2.3886	0.4943	-
	MTR++	TPAMI'24	0.2398	0.8859	1.9712	**0.4106**	-
	RetroMotion (SMoE) (ours)		**0.2515**	**0.8718**	**1.9383**	0.4216	0.1994
	RetroMotion (marginal as joint) (ours)		0.1797	0.8864	2.0118	0.4545	0.2200

Table 5.1: Interactive motion forecasting results of RetroMotion. All methods are evaluated on the interactive splits of the Waymo dataset (Ettinger et al. 2021). The main challenge metric is the mAP. Best scores are **bold**, second best are underlined. Adapted from Wagner et al. (2025).

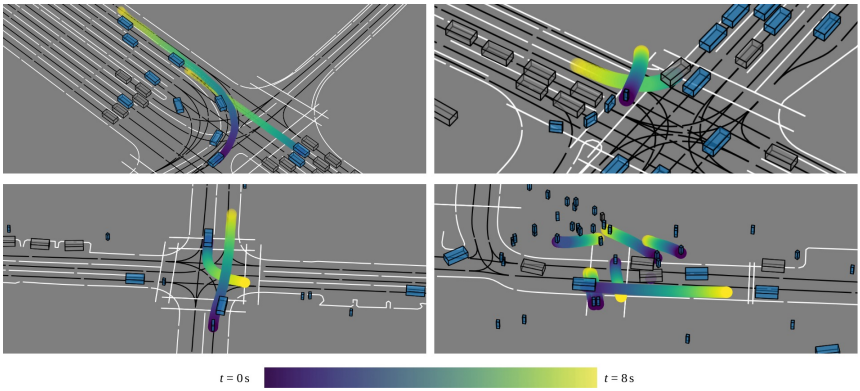


Figure 5.3: Joint and combined motion forecasts of our model. Dynamic agents are shown in blue, static agents in grey (determined at $t = 0\,\mathrm{s}$). Lanes are black lines and road markings are white lines. Top left: forecasts for two cars, top right: a car yielding to a pedestrian on a crosswalk, bottom left: forecasts for a car and a cyclist, and bottom right: combined forecasts of two cars and six pedestrians. Adapted from Wagner et al. (2025).

common agent types, while using larger models and more anchors with fixed initialization seems to improve the results for cyclists. The comparably worse results for cyclists indicate that our method would significantly improve with

more training samples, as in the $1000\times$ larger internal dataset mentioned in (Luo et al. 2023). On average, our method outperforms related methods that require more active parameters.

Method	Params	Anchor	Anchors	Vehicle		Pedestrian		Cyclist	
	(active)	init.		mAP	% of scenes	mAP	% of scenes	mAP	% of scenes
MTR++	87M	Fixed	64	0.3303	100%	0.2088	57%	<u>0.1587</u>	16%
QCNeXt		Learned	6	<u>0.3341</u>		<u>0.2346</u>		0.1369	
BeTopNet	45M	Fixed	64	0.3308		0.2212		**0.1717**	
RetroMotion	24M	Learned	30	**0.3348**		**0.2636**		0.1284	

Table 5.2: Per agent type comparison of the top-4 methods. All metrics are for the interactive test split of the Waymo Open Motion dataset. Params (active) gives the number of parameters that are active per agent (cf. Jiang et al. (2024)). % of scenes gives the percentage of scenes that contain at least one instance of the corresponding agent type. Best scores are **bold**, second best are <u>underlined</u>. Adapted from Wagner et al. (2025).

Our RetroMotion (SMoE hybrid) configuration in Table 5.1 is motivated by the fact that models with more anchors and static anchor initialization perform better for cyclists. Specifically, we build a SMoE model that uses RetroMotion-based experts for vehicles and pedestrians, along with a reproduced BeTopNet model for cyclists.

5.2.2 Cross-dataset generalization

We follow the cross-dataset evaluation protocol of UniTraj (Feng et al. 2024) to measure the generalization capabilities of our method. Specifically, we configure our model to forecasts trajectories of up to 6 seconds and otherwise train the model with the same settings as in Section 5.2.1 and Section 5.2.1.2. Afterwards, we evaluate this model, trained on the Waymo dataset, on the Argoverse 2 dataset.

Table 5.3 shows the results of this experiment. For vehicle trajectories, our method outperforms MTR (Shi et al. 2022) and Wayformer (Nayakanti et al. 2023) and competes with AutoBot (Girgis et al. 2022) (lower miss rate, but higher minFDE scores). For pedestrian trajectories, our method achieves lower minFDE scores,

but a higher miss rate than AutoBot. Overall, these results show that RetroMotion generalizes well and reinforce the results in Table 5.1.

Agent type	Method	Brier-minFDE ↓	minFDE ↓	MR ↓
Vehicle	MTR (Shi et al. 2022)	3.63	3.14	0.44
	Wayformer (Nayakanti et al. 2023)	3.60	3.14	0.45
	AutoBot (Girgis et al. 2022)	**3.23**	**2.41**	<u>0.40</u>
	RetroMotion (ours)	<u>3.51</u>	<u>2.84</u>	**0.31**
Pedestrian	AutoBot (Girgis et al. 2022)	<u>2.22</u>	<u>1.51</u>	**0.16**
	RetroMotion (ours)	**1.97**	**1.26**	<u>0.19</u>

Table 5.3: Cross-dataset generalization from Waymo Open Motion to Argoverse 2 Forecasting. All methods are trained on Waymo Open Motion and evaluated on the validation split of the Argoverse 2 Forecasting dataset (Wilson et al. 2023). Best scores are **bold**, second best are <u>underlined</u>. Adapted from Wagner et al. (2025).

5.2.3 Issuing instructions by modifying trajectories

In the following, we test our model's ability to follow goal-based instructions and to adapt basic directional instructions to the scene context. Specifically, we issue instructions by modifying predicted marginal trajectories prior to re-encoding and joint modeling (see Figure 5.2).

5.2.3.1 Goal-based instructions

In this experiment, we use the last second of the ground truth trajectories as goal-based instructions. Specifically, we replace the last second of predicted marginal trajectories with the last second of ground truth trajectories and evaluate the changes in joint trajectories ($\mathcal{P}^{\text{joint}}_{1:T}$ in Figure 5.2). For evaluation, we use the validation split of the Argoverse 2 Forecasting dataset and the metrics described in Section 2.1.4. This evaluation is similar to the *joint-as-marginal* configuration of Ngiam et al. (2022), but with trajectory modifications as goal-based instructions. At inference, we modify either the marginal trajectory with the highest confidence

score or with the lowest minFDE w.r.t. the desired goal positions, or all 6 trajectories per agent.

We evaluate all instruction configurations on the validation split of the AV2F dataset. Table 5.4 presents the results. All instruction configurations improve the distance-based metrics, while modifying all trajectories leads to the most significant improvement (12% lower final displacement error). This shows that modifications to marginal trajectories affect subsequently decoded joint trajectories and can be used to issue goal-based instructions.

Instruction config.	minFDE↓	minADE↓	MR↓
None	2.45	1.25	0.31
Highest confidence	2.41 -1.6%	1.24	0.31
Lowest minFDE	<u>2.28</u> -6.9%	<u>1.19</u>	<u>0.29</u>
All trajectories	**2.16** -11.8%	**1.15**	**0.27**

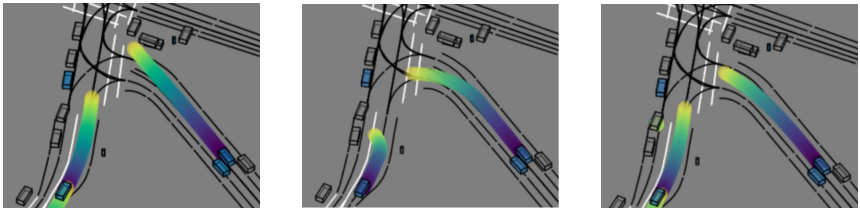
Table 5.4: Goal-based instruction following. As instructions, we modify marginal trajectories and evaluate the changes in joint trajectories. We modify either the trajectory with the highest confidence score or with the lowest minFDE w.r.t. the desired goal positions, or all 6 trajectories. All instruction configurations are evaluated on the validation split of the AV2F dataset. Best scores are **bold**, second best are <u>underlined</u>. Adapted from Wagner et al. (2025).

5.2.3.2 Basic directional instructions

Building on the insight that our model can follow goal-based instructions, we further evaluate basic directional instructions. We define basic directional instructions for turning left and right.

We describe both directional instructions with trajectories based on quarter circles. Specifically, we scale the radius r of the circle to maintain an agent's current speed. Then we use the upper right quadrant, shifted to the origin, as a *turn left* instruction, with $x(t) = r \cdot \cos(t) - r$ and $y(t) = r \cdot \sin(t)$. Similarly, we use the upper left quadrant shifted to the origin as the *turn right* instruction.

At inference, we issue the instructions by replacing the last 4 seconds in all marginal trajectories with the corresponding quarter circle, as shown qualitatively in Figure 5.4 (b).



(a) Predicted marginal trajectories **(b)** *turn left* instruction **(c)** Adapted joint trajectories

Figure 5.4: Our model adapts directional instructions to the scene context. **(a)** shows the default marginal trajectory forecast of our model. **(b)** shows our basic *turn left* instructions, which violate traffic rules by turning into the oncoming lanes. **(c)** shows that our model resolves this by adapting the trajectory of the right vehicle to its lane (shown as black line) and reversing the instruction for the left vehicle, since turning left is not possible. Adapted from Wagner et al. (2025).

However, such instructions are intentionally not adapted to the scene context (map and other agents). In this experiment, we evaluate the ability of RetroMotion to adapt our directional instructions to the given scene context. Specifically, we measure the overlap rate with other agents (see Section 2.1.4) of the modified trajectories used as instructions versus the subsequently decoded joint trajectories. Furthermore, we compute average on-road probability (ORP) scores, which describe the probability of trajectories staying on the road versus going off-road[7].

To compute on-road probability maps, we use a rasterized representation of drivable areas and a distance transform function. The distance transform calculates, for each pixel representing a non-drivable area, the distance to the closest pixel that represents a drivable area. Essentially, it quantifies how far away each non-drivable pixel is from the nearest drivable region. Therefore, this metric also measures how far trajectories leave the drivable area. Figure 5.5 shows an on-road probability map of a scenario in the AV2F dataset. We first compute the on-road

[7] Our metric is related to the drivable area compliance (DAC) metric (Chang et al. 2019).

probability score of each trajectory point. Then, we set a trajectory to off-road if the probability of any trajectory point is < 0.99, and to on-road otherwise. Finally, we compute the average on-road probability (ORP) score as the percentage of on-road trajectories over all trajectories.

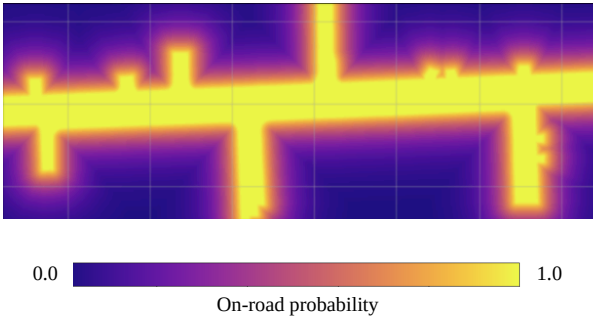


Figure 5.5: On-road probability map of a scenario in the AV2F dataset. The map quantifies how far away each non-drivable pixel is from the nearest drivable area. Adapted from Wagner et al. (2025).

Table 5.5 presents the results of this experiment. Overall, higher ORP scores and lower OR scores are obtained for the *turn right* than for the *turn left* instructions.

Instruction	Eval. trajectory	OR↓	ORP↑
turn left	basic instruction	0.23	0.64
	adapted joint traj.	0.18 -22%	0.85 +33%
turn right	basic instruction	0.21	0.76
	adapted joint traj.	0.18 -14%	0.91 +20%

Table 5.5: Adapting basic directional instructions to the scene context. As instructions, we modify marginal trajectories and evaluate the changes in joint trajectories. In this experiment, we modify all 6 marginal trajectories per agent using our basic directional instructions (cf. Section 5.2.3.2). We report overlap rates (OR) with other agents and on-road probability (ORP) scores. All configurations are evaluated on the validation split of the AV2F dataset. Adapted from Wagner et al. (2025).

We hypothesize that this is due to the fact that the dataset mainly contains right-hand traffic scenarios, where right turns are commonly allowed and more frequent.

Notably, the adjusted joint trajectories have significantly lower OR scores (22% and 14% lower) and much higher ORP scores (33% and 20% higher). This highlights the ability of our model to adapt basic directional instructions to the given scene context.

5.2.4 Analyzing learned trajectory representations

In order to analyze the learned representations and to perform an ablation study on our design choices, we train different configurations of our model. Specifically, we ablate modeling positional uncertainty with different probability distributions, including normal, Laplace, and exponential power distributions. We also train our model with and without compressing the location parameters of the distributions using DCT transforms. We train each model configuration for 14 epochs on the Waymo Open Motion dataset and keep the remaining configurations as in Section 5.2.1 and Section 5.2.1.2.

Table 5.6 presents the results of this experiment. Using Laplace distributions to model positional uncertainty improves motion prediction metrics over using normal distributions. Compressing the location parameters of densities further improves the results significantly. Overall, using exponential power distributions with DCT compression leads to the best results across all metrics.

Distribution	DCT	mAP↑	minFDE↓	minADE↓
Normal	False	0.172	2.177	0.954
Laplace	False	0.176	2.149	0.940
Laplace	True	<u>0.194</u>	<u>2.102</u>	<u>0.917</u>
Exponential power	True	**0.195**	**2.060**	**0.910**

Table 5.6: Comparing distribution types with and without DCT compression. All configurations are evaluated on the interactive validation split of the Waymo Open Motion dataset. Best scores are **bold**, second best are <u>underlined</u>. Adapted from Wagner et al. (2025).

Using DCT compression to compress the location parameters of mixture densities generalizes to Laplace distributions and significantly improves all metrics. Therefore, we evaluate how sensitive our method is to the number of DCT coefficients and train RetroMotion with varying numbers of DCT coefficients.

Table 5.7 shows the results of this experiment. RetroMotion is not too sensitive to the number of DCT coefficients, with our default configuration $N_{\text{DCT}} = 16$ achieving average performance.

N_{DCT}	mAP ↑
12	0.198
16	0.195
20	0.193
24	0.192
28	0.199

Table 5.7: RetroMotion is not too sensitive to N_{DCT}.

5.2.4.1 Weights in exponential power distributions

During training, we track the value of the mixture weights of normal components in exponential power distributions (see Figure 5.6). Initially, the weight is close to 0, because the negative log-likelihood (NLL) of a normal distribution is closely related to the mean squared error, while the NLL of a Laplace distribution is related to the mean absolute error.

$$\text{NLL}_{\text{normal}}(\mu, \sigma) \approx \frac{1}{2\sigma^2} \sum_{n=1}^{N} (x_n - \mu)^2, \tag{5.6}$$

$$\text{NLL}_{\text{Laplace}}(\mu, \sigma) \approx \frac{1}{\sigma} \sum_{n=1}^{N} |x_n - \mu|, \tag{5.7}$$

with predicted mean μ, predicted scale σ, number of samples N, and observed data points x_n. For brevity, we give examples for univariate distributions and without the conditional probabilities described in Equation (5.3) and Equation (5.4).

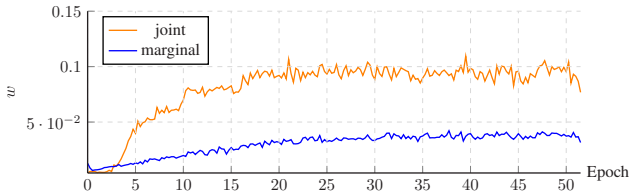


Figure 5.6: Mixture weight of normal components in exponential power distributions. During training the weight w progressively increases, while reaching higher values for joint trajectory distributions than for marginal distributions. Adapted from Wagner et al. (2025).

Therefore, the NLL of a normal distribution is much higher for outliers (i.e., unreasonable predictions during the earlier training epochs), which is compensated by a low w value. During training, w progressively increases, while reaching higher values for predicted joint trajectory distributions than for marginal distributions. However, on average, the learned exponential power distributions tend to be Laplace-like with normal components with relatively low weights of 0.1 (joint) and 0.04 (marginal).

5.2.4.2 Feature vector collapse

We measure the feature vectors collapse metric (NRC1) (Andriopoulos et al. 2024) for feature vectors of marginal and joint trajectory distributions. The feature vectors are the last hidden states of trajectories generated by the MLP heads in our model. The NRC1 metric measures whether d-dimensional feature vectors $\boldsymbol{v}$ collapse to a smaller d_{PCA}-dimensional subspace, where d_{PCA} is equal to the number of principal components.

$$\text{NRC1} = \frac{1}{N} \sum_{n} \left\| \widetilde{\boldsymbol{v}}_n - \text{proj}(\widetilde{\boldsymbol{v}}_n, \boldsymbol{M}_{\text{PCA}}) \right\|_2^2, \tag{5.8}$$

where $\widetilde{\boldsymbol{v}}$ is a normalized feature vector, $\boldsymbol{M}_{\text{PCA}}$ is a $d \times d_{\text{PCA}}$ matrix with columns representing principal components of $\boldsymbol{v}$, and $\text{proj}(\boldsymbol{v}, \boldsymbol{M})$ denotes the projection of $\boldsymbol{v}$ onto the subspace spanned by the columns of $\boldsymbol{M}$.

We compute this metric assuming either 272 or 32 principal components since there are 272 density parameters per trajectory including location parameters compressed as 32 DCT coefficients. Results in Figure 5.7 show an immediate collapse when a 272-dimensional subspace is assumed (upper plot) and no collapse when only considering the first 32 principal components. This observation strongly suggest that the true dimensionality of features vectors lies between 32 and 272, even when considering correlation between output variables, and reinforces the results Table 5.6 and Figure 5.6. These findings support that, in addition to the 32 DCT coefficients for location parameters, other density parameters (i.e., shape and scale) are important when regressing trajectories.

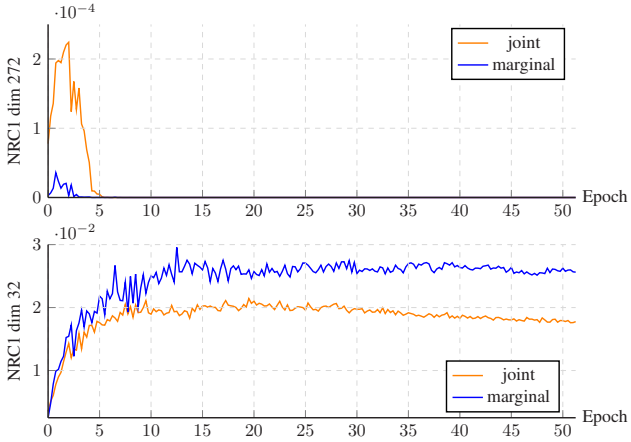


Figure 5.7: Feature vectors collapse (NRC1) for feature vectors of trajectories. The upper plot shows that feature vectors collapses to a subspace spanned by less than 272 principal components. NRC1 does not decrease in the lower plot, indicating that the feature vectors span more than 32 dimensions. Thus, in addition to the 32 DCT coefficients for location parameters, other density parameters are likely important as well. Adapted from Wagner et al. (2025).

5.2.5 Modeling 8 agents jointly

In general, we focus on joint motion forecasting on the interactive splits of the Waymo dataset, which is evaluated in a pairwise joint setting (cf. Ettinger et al. 2021). However, our method is flexible and can be configured to perform joint forecasting for 8 agents. In this section, we extend our experiment in Section 5.2.4 and train RetroMotion to model 8 agents jointly.

Table 5.8 shows the results of this experiment. In comparison to the pairwise joint configuration ($N_{\text{joint}} = 2$) the mAP score is even slightly better, while minFDE and minADE scores are worse. We hypothesize that the version with $N_{\text{joint}} = 8$ focuses more on predicting high probabilities for the joint mode closest to the ground truth, but learns more basic trajectory shapes that are not as precise (worse minFDE and minADE) as there are exponentially more combinations of trajectory shapes when modeling more agents.

N_{joint}	Distribution	DCT	mAP↑	minFDE↓	minADE↓
2	Exponential power	True	0.195	2.060	0.910
8	Exponential power	True	0.196	2.178	0.954

Table 5.8: RetroMotion generalizes to modeling 8 agents jointly. N_{joint} gives the number of jointly modeled agents. Both configurations are evaluated on the interactive validation split of the Waymo dataset.

6 Conclusion

We have explored interpretable representation learning for motion forecasting, including ideas from self-supervised learning, mechanistic interpretability, and mixture density networks. Beyond benchmark-driven model design, we have reverse-engineered *how* forecasting models represent motion, *how* these representations can be modified, and *how* to issue goal-based and directional instructions.

Across the three main contributions, we have shown that:

1. Self-supervised pre-training with interpretable objectives (RedMotion and JointMotion) can improve forecasting accuracy and shorten overall training times. Specifically, RedMotion introduces redundancy reduction objectives for learning environment embeddings from augmented traffic scenes. JointMotion extends this idea to joint motion forecasting with complementary scene-level and instance-level objectives. Both methods improve data efficiency, generalization to new datasets, and balance performance across agent types.

2. Mechanistic interpretability via control vectors (Words in Motion) reveals and modifies the learned representations of human-interpretable motion features such as speed, acceleration, direction, and agent type. Neural collapse analyses show that these features are linearly separable and embedded in a structured latent space. Moreover, control vectors, especially when optimized using sparse autoencoders, enable fine-grained and linear control over motion forecasts at inference. This supports interpetability and enables compensating for domain shifts like different driving styles in new environments.

3. Retrocausal motion forecasting (RetroMotion) leverages a retrocausal information flow from marginal to joint trajectory distributions, supporting accurate interaction modeling and instruction-following. Our model re-encodes marginal forecasts to generate joint distributions for multiple interacting agents, using compressed probabilistic trajectory representations. Notably, without explicit training, our model can follow goal-based instructions and adapt directional instructions to the scene context.

Furthermore, we have performed empirical evaluations on state-of-the-art benchmarks (Waymo Open Motion and Argoverse 2 Forecasting) to validate our methods. RedMotion and JointMotion achieve consistent improvements over strong self-supervised pre-training baselines. Our modifications using control vectors demonstrate interpretability and practical utility, with negligible computational overhead. RetroMotion is instructable and attains competitive or state-of-the-art accuracy on interaction prediction (i.e., pairwise joint motion forecasting).

6.1 Limitations and future work

Our proposed pre-training methods rely more on annotated data than self-supervised pre-training methods used in computer vision (e.g., Chen et al. 2020b, Caron et al. 2021, Bardes et al. 2022). These methods generate training targets from raw images and do not require any annotated data. While our methods do not require labels for motion forecasting, they still rely on map and past trajectory data. Therefore, the data requirements of our methods are more similar to those of self-supervised pre-training of language models (e.g., Radford et al. 2018), which relies on text written by humans.

Moreover, we explored control vectors for categorical features of past motion; extending it to richer, multimodal concepts (e.g., intent or compliance with traffic rules) is a promising direction.

Finally, retrocausal mechanisms, while powerful for controllability, can also enable unrealistic influences between agents. Seff et al. (2023) describe this as

acausal predictions, where, for example, a following vehicle that breaks influences a leading vehicle to break as well. Further investigating and mitigating such artefacts remains important (i.e., analyzing correlation vs. causation).

6.2 Final remarks

We demonstrate that interpretable representation learning is not at odds with state-of-the-art motion forecasting performance. In fact, when approached systematically using control vectors or retrocausal mechanisms, it can enhance model accuracy, robustness, and utility. By integrating pre-training strategies, interpretability analysis, and instructable architectures, our work contributes toward data-driven self-driving systems whose learned representations and mechanisms can be interpreted and controlled.

A Appendix

A.1 Inference latency of our RedMotion model

Model (config)	#agents	Inference latency
RedMotion (mlp-dec)	1	23.2 ms ± 339 µs
	8	25.2 ms ± 642 µs
	64	87.9 ms ± 185 µs
RedMotion (tra-dec)	1	18.2 ms ± 35.9 µs
	8	29.7 ms ± 14.9 µs
	64	146.0 ms ± 41.3 µs

Table A.1: Inference latency of our RedMotion model. Both configurations achieve a low inference latency of less than 30 ms for the challenge setting of forecasting the motion of 8 agents. All times are measured on one A100 GPU using plain PyTorch eager execution and `torch.inference_mode()`. We show mean ± std. dev. of 7 runs with 10 loops each. #agents gives the number of modeled agents. Adapted from Wagner et al. (2024).

A.2 Parameters of our categorical motion features

We classify motion trajectories with a sum less than $15°$ as `straight`. When the cumulative angle exceeds this threshold, a positive value indicates `right` direction, while a negative value – exceeding the threshold in absolute terms – indicates a `left` direction. We classify speeds between $25\,\mathrm{km/h}$ and $50\,\mathrm{km/h}$ as moderate, speeds above this range as `high`, those below as `low`, and negative speeds as `backwards`. For acceleration, we classify trajectories as `decelerating`, if

the integral of speed over time to projected displacement with initial speed is less than 0.9 times. If this ratio is greater than 1.1 times, we classify them as `accelerating`. For all other values, we classify the trajectories as having `constant` speed. We determine all threshold values by analyzing the distribution of the dataset.

A.3 Early, hierarchical and late fusion in motion encoders

We define fusion types for motion transformers based on the information they process in the first attention layers. In early fusion, the first attention layers process motion data of the modeled agent, other agents, and environment context. In hierarchical fusion, they process motion data of the modeled agent, and other agents. In late fusion, they exclusively process motion data of the modeled agent.

A.4 Choosing a range of relative changes in future speed to evaluate control vectors

Given the large range of scenarios in the Waymo dataset, we focus on relative speed changes within a range of $\pm 50\%$ to capture the most relevant speed variations (see Figure 3 in Ettinger et al. (2021)). Considering the approximated mean and standard deviation for each agent type (vehicles: $\mu \approx 12\,\mathrm{m/s}$, $\sigma \approx 5\,\mathrm{m/s}$, pedestrians: $\mu \approx 1.5\,\mathrm{m/s}$, $\sigma \approx 0.7\,\mathrm{m/s}$, and cyclists: $\mu \approx 7\,\mathrm{m/s}$, $\sigma \approx 3\,\mathrm{m/s}$) the $\pm 50\%$ range corresponds to speeds within approximately $\pm 1\sigma$ of the mean for each agent type.

A.5 Evaluating a Koopman autoencoder

A consistent Koopman autoencoder (KoopmanAE) (Azencot et al. 2020) is a bidirectional method that models temporal dynamics between embeddings. The learned latent space approximates a Koopman-invariant space where dynamics evolve linearly. Adapted to the SAE configurations, we train an encoder and a decoder with one layer each and a hidden dimension of 128. Following Azencot et al. (2020), we use learned linear projections to decode Koopman operator approximations $C, D \in \mathbb{R}^{128\times128}$ from intermediate representations. For the first 10 time steps, we encode the embedding and predict the next embedding using C, while for the last 10 time steps, we encode the embedding and predict the previous embedding using D.

Afterwards, we use the KoopmanAE instead of an SAE to fit control vectors (see Section 4.2). Figure A.1 shows the calibration curve for the resulting speed control vector. The range of τ values is approximately $100\times$ smaller than for the SAE-based control vector shown in Figure 4.10b.

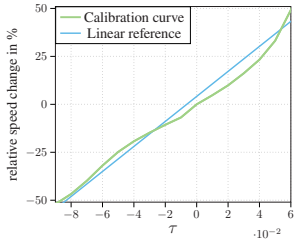


Figure A.1: Calibration curve for a speed control vector optimized using the KoopmanAE. The range of τ values is significantly lower than for plain PCA and SAE-based control vectors (cf. Figure 4.10), yielding lower R^2 scores as shown in Table 4.2. Adapted from Tas and Wagner (2025).

A.6 Fuzz testing PCA-based speed control vectors

Figure A.2 shows the mean jerk and tortuosity values of trajectories when modifying the speed feature with our PCA-based control vector.

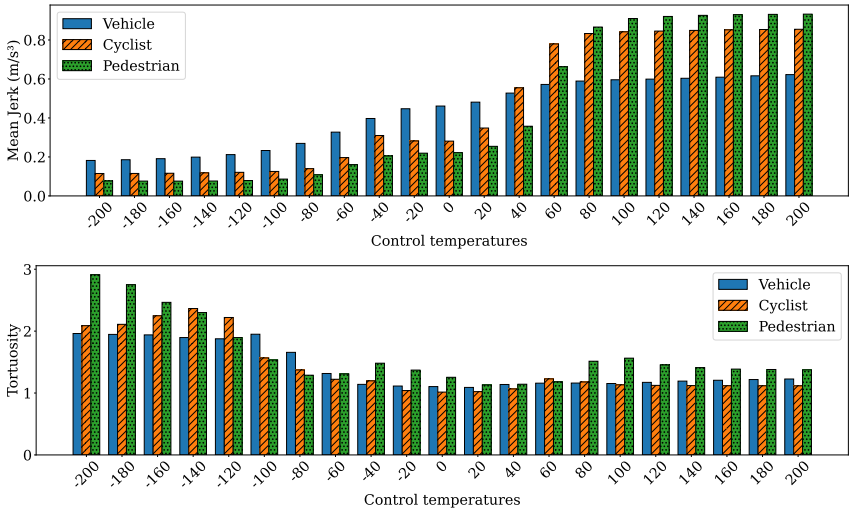


Figure A.2: Fuzz testing control temperatures τ for PCA-based speed control vectors. All metrics are for motion forecasts of our RedMotion model on the Waymo dataset.

A.7 Inference latency when modifying hidden states with control vectors

Table A.2 shows inference latency measurements of a RedMotion model on the Waymo dataset with and without modifying hidden states. Such modifications add only about 1 ms to the total inference latency. Since most datasets are recorded at 10 Hz (e.g., Wilson et al. (2023), Ettinger et al. (2021)), it is common to define the

threshold for real-time processing as $\leq 100\,\mathrm{ms}$. Considering the inference latency of recent 3D detection models (e.g., approx. $40\,\mathrm{ms}$ for DSVT (Wang et al. 2023a)), which must be called before motion forecasting, our modifications should not add significantly to the forecasting latency.

Modifying H	Focal agents	Inference latency
False	8	50.21 ms
True	8	51.08 ms

Table A.2: Inference latency without and with modifying hidden states with our control vectors. We measure the inference latency on one A6000 GPU using the PyTorch Lightning profiler and plain eager execution. We report the mean of 1000 iterations per configuration for the `predict_step`, including pre- and post-processing. Adapted from Tas and Wagner (2025).

A.8 Distance thresholds NMS and softmax τ values for RetroMotion

Expert model	Veh. thresh.	Ped. thresh.	Cyc. thresh.	τ
Vehicle	1.3 m	-	-	0.5
Pedestrian	1.4 m	1.6 m	-	1.1
Cyclist (RetroMotion-based)	1.9 m	-	1.0 m	1.0
Cyclist (BeTopNet-based)	2.5 m	-	2.5 m	-

Table A.3: Distance thresholds NMS and softmax τ values for RetroMotion

List of Figures

List of Tables

List of publications

Journal articles

Royden Wagner, Omer Sahin Tas, Marvin Klemp, Carlos Fernandez, and Christoph Stiller. RedMotion: Motion Prediction via Redundancy Reduction. *Transactions on Machine Learning Research (TMLR)*, 2024. ISSN 2835-8856.

Conference contributions and preprints

Omer Sahin Tas and Royden Wagner. Words in Motion: Extracting Interpretable Control Vectors for Motion Transformers. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Representation Learning (ICLR)*, volume 2025, pages 87191–87214, 2025.

Royden Wagner, Omer Sahin Tas, Marvin Klemp, and Carlos Fernandez. JointMotion: Joint Self-Supervision for Joint Motion Prediction. In *Conference on Robot Learning (CoRL)*, pages 3395–3406. PMLR, 2024a.

Royden Wagner, Ömer Şahin Taş, Marlon Steiner, Fabian Konstantinidis, Hendrik Konigshof, Marvin Klemp, Carlos Fernandez, and Christoph Stiller. SceneMotion: From Agent-Centric Embeddings to Scene-Wide Forecasts. In *2024 IEEE 27th International Conference on Intelligent Transportation Systems (ITSC)*, pages 812–818. IEEE, 2024b.

Marvin Klemp, Royden Wagner, Kevin Rösch, Martin Lauer, and Christoph Stiller. Vehicle Intention Classification using Visual Clues. In *2024 IEEE*

International Conference on Robotics and Automation (ICRA), pages 16395–16401. IEEE, 2024.

Marvin Klemp, Shengyi Chen, Royden Wagner, and Martin Lauer. End-to-End Trainable Deep Neural Network for Radar Interference Detection and Mitigation. In *2023 IEEE International Radar Conference (RADAR)*, pages 1–6. IEEE, 2023a.

Marvin Klemp, Kevin Rösch, Royden Wagner, Jannik Quehl, and Martin Lauer. LDFA: Latent Diffusion Face Anonymization for Self-Driving Applications. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3199–3205, 2023b.

Royden Wagner, Marvin Klemp, Carlos Fernandez Lopez, and Omer Sahin Tas. Road Barlow Twins: Redundancy Reduction for Motion Prediction. In *ICRA2023 Workshop on Pretraining for Robotics (PT4R)*, 2023a.

Royden Wagner, Marvin Klemp, and Carlos Fernandez Lopez. MaskedFusion360: Reconstruct LiDAR Data by Querying Camera Features. In *The First Tiny Papers Track at ICLR 2023, Tiny Papers @ ICLR 2023, Kigali, Rwanda, May 5, 2023. Ed.: K. Maughan Hrsg.: Maughan, Krystal; Liu, Rosanne; Burns, Thomas F.*, 2023b.

Royden Wagner, Omer Sahin Tas, Felix Hauser, Marlon Steiner, Dominik Strutz, Abhishek Vivekanandan, Carlos Fernandez, and Christoph Stiller. RetroMotion: Retrocausal Motion Forecasting Models are Instructable. *arXiv preprint arXiv:2505.20414*, 2025.

Bibliography

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In *International Conference on Learning Representations (ICLR)*, 2017.

George Andriopoulos, Zixuan Dong, Li Guo, Zifan Zhao, and Keith Ross. The prevalence of neural collapse in neural multivariate regression. *Advances in Neural Information Processing Systems (NeurIPS)*, 37, 2024.

Omri Azencot, N Benjamin Erichson, Vanessa Lin, and Michael Mahoney. Forecasting sequential data using consistent koopman autoencoders. In *International Conference on Learning Representations (ICLR)*. PMLR, 2020.

Inhwan Bae, Jean Oh, and Hae-Gon Jeon. Eigentrajectory: Low-rank descriptors for multi-modal trajectory forecasting. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, et al. A cookbook of self-supervised learning. *arXiv preprint arXiv:2304.12210*, 2023.

Mustafa Baniodeh, Kratarth Goel, Scott Ettinger, Carlos Fuertes, Ari Seff, Tim Shen, Cole Gulino, Chenjie Yang, Ghassen Jerfel, Dokook Choe, et al. Scaling laws of motion forecasting and planning–a technical report. *arXiv preprint arXiv:2506.08228*, 2025.

Federico Barbero, Andrea Banino, Steven Kapturowski, Dharshan Kumaran, João Madeira Araújo, Oleksandr Vitvitskyi, Razvan Pascanu, and Petar Veličković.

Transformers need glasses! information over-squashing in language tasks. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:98111–98142, 2024.

Adrien Bardes, Jean Ponce, and Yann Lecun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. In *International Conference on Learning Representations (ICLR)*, 2022.

Horace Barlow. Redundancy reductionrevisited. *Network: computation in neural systems*, 12(3):241, 2001.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Ido Ben-Shaul, Ravid Shwartz-Ziv, Tomer Galanti, Shai Dekel, and Yann LeCun. Reverse engineering self-supervised learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:58324–58345, 2023.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Recognition and Machine Intelligence (PAMI)*, 35(8):1798–1828, 2013.

Simon Benhamou. How to reliably estimate the tortuosity of an animal's path: straightness, sinuosity, or fractal dimension? *Journal of Theoretical Biology*, 2004.

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

Mohamed-Khalil Bouzidi, Bojan Derajic, Daniel Goehring, and Joerg Reichardt. Motion planning under uncertainty: Integrating learning-based multi-modal predictors into branch model predictive control. *arXiv preprint arXiv:2405.03470*, 2024.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas

Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1877–1901, 2020.

Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9650–9660, 2021.

Sergio Casas, Cole Gulino, Simon Suo, Katie Luo, Renjie Liao, and Raquel Urtasun. Implicit latent variable model for scene-consistent motion forecasting. In *European Conference on Computer Vision (ECCV)*, pages 624–641. Springer, 2020.

Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. In *Conference on Robot Learning (CoRL)*. PMLR, 2020.

Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8748–8757, 2019.

Dian Chen, Vladlen Koltun, and Philipp Krähenbühl. Learning To Drive From a World on Rails. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021a.

Hao Chen, Jiaze Wang, Kun Shao, Furui Liu, Jianye Hao, Chenyong Guan, Guangyong Chen, and Pheng-Ann Heng. Traj-mae: Masked autoencoders for trajectory prediction. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8351–8362, 2023.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:15084–15097, 2021b.

Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International Conference on Machine Learning (ICML)*, pages 1691–1703. PMLR, 2020a.

Shizhe Chen, Pierre-Louis Guhur, Cordelia Schmid, and Ivan Laptev. History aware multimodal transformer for vision-and-language navigation. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021c.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning (ICML)*, pages 1597–1607. PmLR, 2020b.

Xinlei Chen and Kaiming He. Exploring Simple Siamese Representation Learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Jie Cheng, Xiaodong Mei, and Ming Liu. Forecast-mae: Self-supervised pretraining for motion forecasting with masked autoencoders. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *IEEE/CVF Conference*

on Computer Vision and Pattern Recognition (CVPR), volume 1, pages 539–546. IEEE, 2005.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: Reverse engineering how networks learn group operations. In *International Conference on Machine Learning (ICML)*, pages 6243–6267. PMLR, 2023.

Alexander Cui, Sergio Casas, Kelvin Wong, Simon Suo, and Raquel Urtasun. Gorela: Go relative for viewpoint-invariant motion forecasting. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Annual Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*, pages 4171–4186, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, G Heigold, S Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles R Qi, Yin Zhou, et al. Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9710–9719, 2021.

William Fedus, Jeff Dean, and Barret Zoph. A review of sparse expert models in deep learning. *arXiv preprint arXiv:2209.01667*, 2022.

Christoph Feichtenhofer, Yanghao Li, Kaiming He, et al. Masked autoencoders as spatiotemporal learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:35946–35958, 2022.

Lan Feng, Mohammadhossein Bahari, Kaouther Messaoud Ben Amor, Éloi Zablocki, Matthieu Cord, and Alexandre Alahi. Unitraj: A unified framework for scalable vehicle trajectory prediction. In *European Conference on Computer Vision (ECCV)*, pages 106–123. Springer, 2024.

Lan Feng, Mohammadhossein Bahari, Kaouther Messaoud Ben Amor, Éloi Zablocki, Matthieu Cord, and Alexandre Alahi. Unitraj: A unified framework for scalable vehicle trajectory prediction. In *European Conference on Computer Vision (ECCV)*, 2025.

Enrico Fini, Pietro Astolfi, Adriana Romero-Soriano, Jakob Verbeek, and Michal Drozdzal. Improved baselines for vision-language pre-training. *Transactions on Machine Learning Research (TMLR)*, 2023.

Tomer Galanti, András György, and Marcus Hutter. On the role of neural collapse in transfer learning. *arXiv preprint arXiv:2112.15121*, 2021.

Mudasir A Ganaie, Minghui Hu, Ashwani Kumar Malik, Muhammad Tanveer, and Ponnuthurai N Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, 2022.

Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *International Conference on Learning Representations (ICLR)*, 2025.

Maximilian Geisslinger, Franziska Poszler, and Markus Lienkamp. An ethical trajectory planning algorithm for autonomous vehicles. *Nature Machine Intelligence*, 2023.

Thomas Gilles, Stefano Sabatini, Dzmitry Tsishkou, Bogdan Stanciulescu, and Fabien Moutarde. Thomas: Trajectory heatmap output with learned multi-agent sampling. In *International Conference on Learning Representations (ICLR)*, 2022.

Roger Girgis, Florian Golemo, Felipe Codevilla, Martin Weiss, Jim Aldon D'Souza, Samira E Kahou, Felix Heide, and Christopher Pal. Latent variable sequential set transformers for joint multi-agent motion prediction. In *International Conference on Learning Representations (ICLR)*, 2022.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:21271–21284, 2020.

Pierre-Louis Guhur, Shizhe Chen, Ricardo Garcia Pinel, Makarand Tapaswi, Ivan Laptev, and Cordelia Schmid. Instruction-driven history-aware policies for robotic manipulations. In *Conference on Robot Learning (CoRL)*, pages 175–187. PMLR, 2023.

Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing. *Transactions on Machine Learning Research (TMLR)*, 2023. ISSN 2835-8856.

Abner Guzman-Rivera, Dhruv Batra, and Pushmeet Kohli. Multiple choice learning: Learning to produce multiple structured outputs. *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9729–9738, 2020.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16000–16009, 2022.

John Houston, Guido Zuidhof, Luca Bergamini, Yawei Ye, Long Chen, Ashesh Jain, Sammy Omari, Vladimir Iglovikov, and Peter Ondruska. One thousand and one hours: Self-driving motion prediction dataset. In *Conference on Robot Learning (CoRL)*, 2021.

Zhiyu Huang, Haochen Liu, and Chen Lv. Gameformer: Game-theoretic modeling and learning of transformer-based interactive prediction and planning for autonomous driving. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3903–3913, 2023.

Ronny Hug, Wolfgang Hübner, and Michael Arens. Introducing probabilistic bézier curves for n-step sequence prediction. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Masha Itkina and Mykel Kochenderfer. Interpretable self-aware neural networks for robust trajectory prediction. In *Conference on Robot Learning (CoRL)*. PMLR, 2023.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning (ICML)*, 2021.

Xiaosong Jia, Penghao Wu, Li Chen, Yu Liu, Hongyang Li, and Junchi Yan. HDGT: Heterogeneous Driving Graph Transformer for Multi-Agent Trajectory

Prediction via Scene Encoding. *IEEE Transactions on Pattern Recognition and Machine Intelligence (PAMI)*, 2023.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Chiyu Jiang, Andre Cornman, Cheolho Park, Benjamin Sapp, Yin Zhou, Dragomir Anguelov, et al. Motiondiffuser: Controllable multi-agent motion prediction using diffusion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

William Johnson and Joram Lindenstrauss. Extensions of Lipschitz maps into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 01 1984. doi: 10.1090/conm/026/737400.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 33: 18661–18673, 2020.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, et al. Openvla: An open-source vision-language-action model. In *Conference on Robot Learning (CoRL)*, 2024.

Ouail Kitouni, Niklas Nolte, Vıctor Samuel Pérez-Dıaz, Sokratis Trifinopoulos, and Mike Williams. From neurons to neutrons: A case study in interpretability. In *International Conference on Machine Learning (ICML)*, pages 24726–24748. PMLR, 2024.

Elyor Kodirov, Tao Xiang, Zhenyong Fu, and Shaogang Gong. Unsupervised domain adaptation for zero-shot learning. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2015.

Stepan Konev. Mpa: Multipath++ based architecture for motion prediction. *arXiv preprint arXiv:2206.10041*, 2022.

Stepan Konev, Kirill Brodt, and Artsiom Sanakoyeu. Motioncnn: A strong baseline for motion prediction in autonomous driving. *arXiv preprint arXiv:2206.02163*, 2022.

Sébastien Lachapelle, Divyat Mahajan, Ioannis Mitliagkas, and Simon Lacoste-Julien. Additive decoders for latent variables identification and cartesian-product extrapolation. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2024.

Zhiqian Lan, Yuxuan Jiang, Yao Mu, Chen Chen, and Shengbo Eben Li. SEPT: Towards efficient scene representation learning for motion prediction. In *International Conference on Learning Representations (ICLR)*, 2024.

Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Ng. Efficient sparse coding algorithms. *Advances in Neural Information Processing Systems (NeurIPS)*, 19, 2006.

Stefan Lee, Senthil Purushwalkam Shiva Prakash, Michael Cogswell, Viresh Ranjan, David Crandall, and Dhruv Batra. Stochastic multiple choice learning for training diverse deep ensembles. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Janet Levin. Functionalism. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2023 edition, 2023.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

Haochen Liu, Li Chen, Yu Qiao, Chen Lv, and Hongyang Li. Reasoning multi-agent behavioral topology for interactive autonomous driving. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Wenjie Luo, Cheol Park, Andre Cornman, Benjamin Sapp, and Dragomir Anguelov. Jfp: Joint future prediction with interactive multi-agent modeling for autonomous driving. In *Conference on Robot Learning (CoRL)*, 2023.

Osama Makansi, Eddy Ilg, Ozgun Cicek, and Thomas Brox. Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7144–7153, 2019.

Karttikeya Mangalam, Harshayu Girase, Shreyas Agarwal, Kuan-Hui Lee, Ehsan Adeli, Jitendra Malik, and Adrien Gaidon. It is not the journey but the destination: Endpoint conditioned trajectory prediction. In *European Conference on Computer Vision (ECCV)*, pages 759–776. Springer, 2020.

Wei Mao, Miaomiao Liu, Mathieu Salzmann, and Hongdong Li. Learning trajectory dependencies for human motion prediction. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9489–9497, 2019.

Evan Markou, Thalaiyasingam Ajanthan, and Stephen Gould. Guiding neural collapse: Optimising towards the nearest simplex equiangular tight frame. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:35544–35573, 2024.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:17359–17372, 2022.

Marco Mistretta, Alberto Baldrati, Marco Bertini, and Andrew D Bagdanov. Improving zero-shot generalization of learned prompts via unsupervised knowledge distillation. In *European Conference on Computer Vision (ECCV)*, 2024.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations (ICLR)*, 2023.

Nigamaa Nayakanti, Rami Al-Rfou, Aurick Zhou, Kratarth Goel, Khaled S Refaat, and Benjamin Sapp. Wayformer: Motion forecasting via simple & efficient attention networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2980–2987. IEEE, 2023.

Jiquan Ngiam, Vijay Vasudevan, Benjamin Caine, Zhengdong Zhang, Hao-Tien Lewis Chiang, Jeffrey Ling, Rebecca Roelofs, Alex Bewley, Chenxi Liu, Ashish Venugopal, et al. Scene transformer: A unified architecture for predicting future trajectories of multiple agents. In *International Conference on Learning Representations (ICLR)*, 2022.

Toan Q Nguyen and Julian Salazar. Transformers without tears: Improving the normalization of self-attention. In *International Conference on Spoken Language Translation*, 2019.

Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *Transactions on Machine Learning Research (TMLR)*, 2024.

Vardan Papyan, XY Han, and David L Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.

Alexander Pashevich, Cordelia Schmid, and Chen Sun. Episodic transformer for vision-and-language navigation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Maria Abi Raad, Arun Ahuja, Catarina Barros, Frederic Besse, Andrew Bolt, Adrian Bolton, Bethanie Brownfield, Gavin Buttimore, Max Cant, Sarah Chakera, et al. Scaling instructable agents across many simulated worlds. *arXiv preprint arXiv:2404.10179*, 2024.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI Blog*, 2018.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, pages 8748–8763. PMLR, 2021.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning (ICML)*, pages 28492–28518. PMLR, 2023.

Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *Advances in Neural Information Processing Systems (NeurIPS)*, 34: 12116–12128, 2021.

Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. Jumping Ahead: Improving Reconstruction Fidelity with JumpReLU Sparse Autoencoders. *arXiv:2407.14435*, 2024.

Sucheng Ren, Zeyu Wang, Hongru Zhu, Junfei Xiao, Alan Yuille, and Cihang Xie. Rejuvenating image-gpt as strong visual representation learners. In *International Conference on Machine Learning (ICML)*, pages 42449–42461, 2024.

Gabriel Sarch, Sahil Somani, Raghav Kapoor, Michael J Tarr, and Katerina Fragkiadaki. Helper-x: A unified instructable embodied agent to tackle four interactive vision-language domains with memory-augmented language models. *arXiv preprint arXiv:2404.19065*, 2024.

Steffen Schneider, Jin Hwa Lee, and Mackenzie Weygandt Mathis. Learnable latent embeddings for joint behavioural and neural analysis. *Nature*, 617(7960): 360–368, 2023.

Ari Seff, Brian Cera, Dian Chen, Mason Ng, Aurick Zhou, Nigamaa Nayakanti, Khaled S Refaat, Rami Al-Rfou, and Benjamin Sapp. Motionlm: Multi-agent motion forecasting as language modeling. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. Motion transformer with global intention localization and local movement refinement. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:6531–6543, 2022.

Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. Mtr++: Multi-agent motion prediction with symmetric scene modeling and guided intention querying. *IEEE Transactions on Pattern Recognition and Machine Intelligence (PAMI)*, 2024.

DiJia Andy Su, Bertrand Douillard, Rami Al-Rfou, Cheol Park, and Benjamin Sapp. Narrowing the coordinate-frame gap in behavior prediction models: Distillation for efficient and accurate scene-centric motion forecasting. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 653–659. IEEE, 2022.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomputing*, 568, 2024.

Jiawei Sun, Jiahui Li, Tingchen Liu, Chengran Yuan, Shuo Sun, Zefan Huang, Anthony Wong, Keng Peng Tee, and Marcelo H Ang Jr. Rmp-yolo: A robust motion predictor for partially observable scenarios even if you only look once. *arXiv preprint arXiv:2409.11696*, 2024a.

Qiao Sun, Xin Huang, Junru Gu, Brian C Williams, and Hang Zhao. M2i: From factored marginal trajectory prediction to interactive prediction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Zhiqing Sun, Yikang Shen, Hongxin Zhang, Qinhong Zhou, Zhenfang Chen, David Daniel Cox, Yiming Yang, and Chuang Gan. Salmon: Self-alignment with instructable reward models. In *International Conference on Learning Representations (ICLR)*, 2024b.

Ömer Şahin Taş, Philipp Heinrich Brusius, and Christoph Stiller. Decision-theoretic mpc: Motion planning with weighted maneuver preferences under uncertainty. *arXiv preprint arXiv:2310.17963*, 2023.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024. URL `https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html`.

Ekaterina Tolstaya, Reza Mahjourian, Carlton Downey, Balakrishnan Vadarajan, Benjamin Sapp, and Dragomir Anguelov. Identifying driver interactions via conditional behavior prediction. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3473–3479. IEEE, 2021.

Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. MLP-Mixer: An all-MLP Architecture for Vision. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning (ICML)*, pages 1747–1756. PMLR, 2016.

Balakrishnan Varadarajan, Ahmed Hefny, Avikalp Srivastava, Khaled S Refaat, Nigamaa Nayakanti, Andre Cornman, Kan Chen, Bertrand Douillard, Chi Pang Lam, Dragomir Anguelov, et al. Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7814–7821. IEEE, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.

Haiyang Wang, Chen Shi, Shaoshuai Shi, Meng Lei, Sen Wang, Di He, Bernt Schiele, and Liwei Wang. DSVT: Dynamic Sparse Voxel Transformer With Rotated Sets. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023a.

Xishun Wang, Tong Su, Fang Da, and Xiaodong Yang. Prophnet: Efficient agent-centric motion forecasting with anchor-informed proposals. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21995–22003, 2023b.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, et al. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. *arXiv preprint arXiv:2412.13663*, 2024.

Marissa A Weis, Laura Pede, Timo Lüddecke, and Alexander S Ecker. Self-supervised graph representation learning for neuronal morphologies. *Transactions on Machine Learning Research (TMLR)*, 2023.

Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, et al. Argoverse 2: Next generation datasets for self-driving perception and forecasting. *arXiv preprint arXiv:2301.00493*, 2023.

Florian Wirth. *Conditional Behavior Prediction of Interacting Agents on Map Graphs with Neural Networks*. PhD thesis, Karlsruhe Institute of Technology (KIT), 2023.

Robert Wu and Vardan Papyan. Linguistic collapse: Neural collapse in (large) language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:137432–137473, 2024.

Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-Shot Learning - the Good, the Bad and the Ugly. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Chenfeng Xu, Tian Li, Chen Tang, Lingfeng Sun, Kurt Keutzer, Masayoshi Tomizuka, Alireza Fathi, and Wei Zhan. Pretram: Self-supervised pre-training via connecting trajectory and map. In *European Conference on Computer Vision (ECCV)*, pages 34–50. Springer, 2022.

Yihong Xu, Victor Letzelter, Mickaël Chen, Éloi Zablocki, and Matthieu Cord. Annealed winner-takes-all for motion forecasting. *arXiv preprint arXiv:2409.11172*, 2024.

Yi Yang, Qingwen Zhang, Thomas Gilles, Nazre Batool, and John Folkesson. Rmp: A random mask pretrain framework for motion prediction. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 3717–3723. IEEE, 2023.

Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning (ICML)*, pages 12310–12320. PMLR, 2021.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12104–12113, 2022.

Zhejun Zhang, Alexander Liniger, Christos Sakaridis, Fisher Yu, and Luc V Gool. Real-time motion prediction via heterogeneous polyline transformer with relative pose encoding. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Zikang Zhou, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. Query-centric trajectory prediction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17863–17873, 2023a.

Zikang Zhou, Zihao Wen, Jianping Wang, Yung-Hui Li, and Yu-Kai Huang. Qcnext: A next-generation framework for joint multi-agent trajectory prediction. *arXiv preprint arXiv:2306.10508*, 2023b.

Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization. *arXiv preprint arXiv:2503.10622*, 2025.

Zhihui Zhu, Tianyu Ding, Jinxin Zhou, Xiao Li, Chong You, Jeremias Sulam, and Qing Qu. A geometric analysis of neural collapse with unconstrained features. *Advances in Neural Information Processing Systems (NeurIPS)*, 34: 29820–29834, 2021.

Hans-Georg Zimmermann, Christoph Tietz, and Ralph Grothmann. Forecasting with recurrent neural networks: 12 tricks. *Neural Networks: Tricks of the Trade: Second Edition*, pages 687–707, 2012.

Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin Dogus Cubuk, and Quoc Le. Rethinking pre-training and self-training. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation Engineering: A Top-Down Approach to AI Transparency. *arXiv:2310.01405*, 2023.