WORDS IN MOTION: EXTRACTING INTERPRETABLE CONTROL VECTORS FOR MOTION TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer-based models generate hidden states that are difficult to interpret. In this work, we aim to interpret these hidden states and control them at inference, with a focus on motion forecasting. We leverage the phenomenon of neural collapse and use linear probes to measure interpretable features in hidden states. Our experiments reveal meaningful directions and distances between hidden states of opposing features, which we use to fit control vectors for activation steering. We further refine our approach using sparse autoencoders to optimize our control vectors. Notably, we show that enforcing sparsity leads to a more linear relationship between control vector temperatures and forecasts. Our approach not only enables mechanistic interpretability but also zero-shot generalization to unseen dataset characteristics.[1]

1 INTRODUCTION

Accurately predicting sequential data while maintaining interpretability is crucial for many real-world applications. However, these two objectives often conflict, as achieving higher accuracy frequently comes at the cost of lower interpretability. This trade-off is primarily linked to the representational capacity of the underlying model: methods achieving higher accuracy tend to rely on the increased complexity of their underlying models (Kaplan et al., 2020; Bahri et al., 2024). This, in turn, renders them difficult to understand and interpret in terms of semantically meaningful patterns.

Deep learning models employ loss functions that encourage the clustering of data samples based on their patterns (Papyan et al., 2020). Together with regularizers that prevent overfitting, clusters become more distinct over the course of training, i.e. *neural collapse* (Galanti et al., 2022; Wu & Papyan, 2024). We leverage this phenomenon to analyze the learned representations of transformer-based models with respect to human-interpretable features during training. Specifically, we use linear probing (Alain & Bengio, 2017) to measure the degree to which these features are embedded in hidden states. In this way, we identify that interpretable features are embedded in the hidden states of transformer-based models.

Building on these insights, we fit control vectors to opposing features, enabling the control of forecasts at inference. To further enhance this approach, we employ sparse autoencoders to extract more distinct features from hidden states (Bricken et al., 2023). Experiments with sparse autoencoders of varying hidden dimensions reveal that enforcing sparsity leads to a more linear relationship between control vector temperatures and the resulting forecasts. Consequently, our method allows for controlling transformer-based forecasting models through interpretable control vectors, providing a novel and intuitive interface that facilitates zero-shot generalization.

Our application focuses on recent multimodal transformer-based motion forecasting models (Nayakanti et al., 2023; Zhang et al., 2023b; Wagner et al., 2024). They process features of past motion sequences (i.e., past positions, orientation, acceleration, speed) and environment context (i.e., map data and traffic light states), and transform them into future motion sequences. Like other transformer models, they rely on learned representations of these features, resulting in hidden states that are difficult to interpret and control.

Our contributions are:

[1] We plan to make our software implementation and trained models publicly available.

- We use linear probes to measure neural collapse towards interpretable features in hidden states of recent motion transformer models. We show that the collapse of hidden states creates meaningful directions and distances in latent space.
- We leverage these latent space properties to fit control vectors for each interpretable feature. We optimize our control vectors with sparse autoencoding. Notably, we show that enforcing sparsity leads to a more linear relationship between control vector temperatures and forecasts.
- We apply our method to motion forecasting models with various fusion mechanisms and environment representations. Furthermore, we address domain shift using our interpretable control vectors and enable zero-shot generalization.

2 Related Work

2.1 Natural language as an interface for model interaction and control

Linking learned representations to natural language has gained significant attention (e.g., (Radford et al., 2021; Alayrac et al., 2022; Liu et al., 2024)). Broadly, approaches incorporating language in models can be categorized into four types.

Conditioning. Numerous works leverage natural language to condition generative models in diverse tasks such as image synthesis (Ramesh et al., 2021; Zhang et al., 2023a), video generation (Blattmann et al., 2023), and 3D modeling (Tevet et al., 2022; Wu et al., 2023).

Prompting. Some works use language as an interface to interact with models, enabling users to request assistance or information. This includes obtaining explanations of underlying reasoning, and human-centric descriptions of model behavior (Brown et al., 2020; Sanh et al., 2021).

Enriching. Another line of work leverages LLMs' generalization abilities to enrich context embeddings, providing additional information for better prediction and planning (Guan et al., 2023).

Instructing. Natural language can be used to issue explicit commands for specific tasks, distinct from conditioning (Ouyang et al., 2022; Brooks et al., 2023). The main challenge is connecting the abstractions and generality of language with environment-grounded actions (Raad et al., 2024).

In the appendix, we provide further examples of these language incorporation approaches in robotics and self-driving applications. While these works align learned text representations with embeddings of other modalities, they do not measure the degree to which interpretable features are embedded within hidden states. To the best of our knowledge, no prior work has explored the mechanistic interpretability of transformers in robotics applications.

2.2 Methods for model interpretability and explainability

Latent space regularities. Mikolov et al. (2013) show that consistent regularities naturally emerge from the training process of word embeddings. This phenomenon, commonly referred to as the "word2vec hypothesis", suggests that learned embeddings capture both semantic and syntactic relationships between words through consistent vector offsets in latent space. Many multimodal models, including CLIP (Radford et al., 2021), use contrastive learning to align embeddings across different modalities and maximize their cosine similarity. It should be emphasized that these works measure the similarity of embeddings of all features per sample, rather than focusing on distinct features within the latent space.

Neural collapse. A recent line of work (Papyan et al., 2020; Galanti et al., 2022; Wu & Papyan, 2024) introduces the term *neural collapse* to describe a desirable learning behavior of deep neural networks for classification.[2] It refers to the phenomenon that learned top-layer representations form semantic clusters, which collapse to their means at the end of training (see Appendix A.10 for details). In addition, the cluster means transform progressively into equidistant vectors when centered around the global mean. Therefore, neural collapse facilitates classification tasks and is considered a desirable learning behavior for both supervised (Papyan et al., 2020) and self-supervised learning (Ben-Shaul et al., 2023).

[2]Neural collapse is not to be confused with representation collapse (Hua et al., 2021; Barbero et al., 2024), where learned representations across all classes collapse to redundant or trivial solutions (e.g., zero vectors).

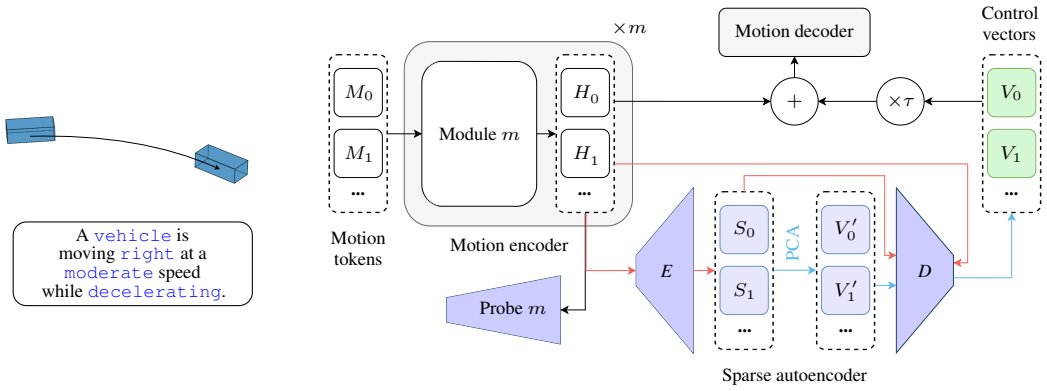
Hidden state activations. Transformers consist of attention blocks, followed by simple feed-forward networks, whose hidden state activations are analyzed for interpretability. Elhage et al. (2022) explore two key hypotheses that describe how these activations capture meaningful structures: the linear representation hypothesis (Pennington et al., 2014) and the superposition hypothesis (Arora et al., 2018). These hypotheses essentially state that the neural networks represent features as directions in their activation space, and that representations can be decomposed into independent features.

Control vectors. In natural language processing (Zou et al., 2023; Subramani et al., 2022; Turner et al., 2023), control vectors allow targeted adjustments to model outputs by steering hidden state activations without the need for fine-tuning or prompt engineering. Control vectors are a set of vectors that capture the difference of hidden states from opposing concepts or features (Rimsky et al., 2023). This approach requires a well-structured latent space, where samples are clustered according to classes or features (e.g., a high degree of neural collapse, see Section 3.2).

Sparse autoencoders. A key goal of interpretability research is to decompose models and gain a mechanistic interpretation of how their components function. Sparse autoencoders leverage the linear representation hypothesis and approximate the model's activations with a linear combination of feature directions. By enforcing sparsity in latent space, they separate features into distinct, interpretable representations (Bricken et al., 2023; Cunningham et al., 2023).

Our method differs from prior works in several aspects. We measure neural collapse in multimodal models for motion forecasting (i.e., regression) instead of unimodal vision classifiers (Papyan et al., 2020) or language models (Wu & Papyan, 2024). Rather than steering hidden state changes across all modules (i.e., neural trajectories) as in Zou et al. (2023), we steer only the hidden states in the last module of the motion encoder. Furthermore, we do not use our sparse autoencoders during inference (Cunningham et al., 2023), but to optimize control vectors beforehand, resulting in negligible computational overhead.

3 METHOD



(a) Interpretable motion features

(b) Controllable motion encoder

Figure 1: **Words in Motion. (a)** We classify motion features in an interpretable way, as in natural language. **(b)** We measure the degree to which these interpretable features are embedded in the hidden states H_i of transformer models with linear probes. Furthermore, we use our discrete features and sparse autoencoding to fit interpretable control vectors V_i that allow for controlling motion forecasts at inference. The training of the sparse autoencoder is shown with red arrows ($\rightarrow$) and the fitting of control vectors with blue arrows ($\rightarrow$).

3.1 MOTION FEATURE CLASSIFICATION USING NATURAL LANGUAGE

In contrast to natural language, where words naturally carry semantic meaning, motion lacks predefined labels. Therefore, we identify human-interpretable motion features by quantizing them into discrete subclasses as in natural language. Our classes of motion features are based on insights from Seff et al. (2023).

Initially, we classify motion direction using the cumulative sum of differences in yaw angles, assigning it to either `left`, `straight`, or `right`. Additionally, we introduce a `stationary` class for stationary objects, where direction lacks semantic significance. We define further classes 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 1a). The threshold values used for classification are detailed in the appendix.

3.2 Neural collapse as a metric of interpretability

We propose to measure neural collapse as a metric of interpretability. Specifically, we focus on interpreting hidden states (i.e., latent representations) and evaluate whether hidden states embed human-interpretable features. We measure how close abstract hidden states are related to interpretable semantics using linear probing accuracy (Alain & Bengio, 2017).[3] We train linear probes (i.e., linear classifiers detached from the overall gradient computation) on top of hidden states (H_i in Figure 1). During training, we track their accuracy in classifying our interpretable features on validation sets. Adapted to motion forecasting, we choose the aforementioned motion features as interpretable semantics.

Besides linear probing accuracy, following Chen & He (2021), we use the mean of the standard deviation of the ℓ_2-normalized embedding to measure representation collapse. Representation collapse refers to an undesirable learning behaviour where learned embeddings collapse into redundant or trivial representations (Hua et al., 2021; Barbero et al., 2024). Redundant representations have a standard deviation close to zero. In a way, representing the opposite of neural collapse. As shown in (Chen & He, 2021), rich representations have a standard deviation close to $1/\sqrt{\text{dim}}$, where dim is the hidden dimension.

3.3 Interpretable control vectors

We use interpretable features to build pairs of opposing features. For each pair, we build a dataset and extract the corresponding hidden states. Afterwards, we compute the element-wise difference between the hidden states of samples with opposing features. Finally, we follow Zou et al. (2023) and use principal component analysis (PCA) with one component as pooling method to reduce the computed differences to one scalar per hidden dimension and to generate control vectors (V_i in Figure 1b).

We optimize our control vectors using sparse autoencoders (Cunningham et al., 2023). The sparse autoencoder is trained as an auxiliary network. It extracts distinct features in hidden states by encoding and reconstructing them from a sparse intermediate representation. We hypothesize that this sparse intermediate representation (S_i in Figure 1b) enables a more linear decomposition of our interpretable features, and hence, more distinct control vectors. Therefore, we use these sparse intermediate representations to generate intermediate control vectors V_i' by pooling the differences of samples with opposing features. Leveraging the Johnson-Lindenstrauss Lemma,[4] we use the SAE decoder to project the intermediate control vectors back to the hidden dimension of the motion encoder. This enables using sparse autoencoders of arbitrary sparse intermediate dimensions for generating control vectors of fixed dimension.

At inference, we scale the control vectors with a temperature parameter (τ in Figure 1b) to control the strength of the corresponding features of a given sample. Consequently, we propose sparse autoencoding to generate control vectors with a more linear relationship between control temperatures and forecasts.

[3] Ben-Shaul et al. (2023) show that linear probing accuracy is consistent with the accuracy of nearest class center classifiers, which are typically used to measure neural collapse.

[4] Johnson & 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.

4 EXPERIMENTAL SETUP

4.1 MOTION FORECASTING MODELS

We study three recent transformer models for motion forecasting in self-driving vehicles. Wayformer (Nayakanti et al., 2023) and RedMotion (Wagner et al., 2024) models employ attention-based scene encoders to learn agent-centric embeddings of past motion, map, and traffic light data. To efficiently process long token sequences, Wayformer uses latent query attention (Jaegle et al., 2021) for subsampling, RedMotion lowers memory requirements via local-attention (Beltagy et al., 2020) and redundancy reduction. HPTR (Zhang et al., 2023b) models learn pairwise-relative environment representations via kNN-based attention mechanisms. For Wayformer, we use the implementation by Zhang et al. (2023b) and the early fusion configuration. Therefore we analyze the hidden states generated by MLP-based input projectors for motion data, which consists of three layers. For RedMotion, we use the publicly available implementation with a late fusion encoder for motion data (Wagner et al., 2024). For HPTR, we use the implementation by Zhang et al. (2023b) and a custom hierarchical fusion setup with a modality-specific encoder for past motion and a shared encoder for environment context. Further details on fusion mechanisms and model architectures are presented in appendix A.5 and A.12.

4.2 LINEAR PROBES

We add linear probes for our quantized motion features to each hidden state of all models ($H_i^{[m]}$ in Figure 1, where $m \in \{0, 1, 2\}$ is the module number and i is the token index). These classifiers are learned during training using regular cross-entropy loss to classify speed, acceleration, direction, and the agent classes from hidden states. We decouple this objective from the overall gradient computation. Therefore, these classifiers do not contribute to the alignment of latent representations, but exclusively measure the corresponding neural collapse into interpretable clusters.

4.3 CONTROL VECTORS

Using our interpretable motion features (see Section 3.1), we build pairs of opposing features. Specifically, we generate *speed* control vectors representing the transition from low to high speed, *acceleration* control vectors representing the transition from decelerating to accelerating, and *direction* control vectors representing the transition from the left and right direction, and *agent* control vectors representing the transition from pedestrian to vehicle (see Section 3.3). For each pair, we use the hidden states $H_i^{[m]}$ from module $m = 2$ and the last token per motion sequence (with $i = -1$), as it is closest to the start of the prediction.

4.4 TRAINING DETAILS AND HYPERPARAMETERS

Motion forecasting transformers. 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. For the Argoverse 2 Forecasting (abbr. AV2F) dataset, we use past motion sequences with 50 timesteps (representing $5\,\mathrm{s}$) as input. For the Waymo Open Motion (abbr. Waymo) dataset, we use past motion sequences with 11 steps (representing $1.1\,\mathrm{s}$) as input. As the main loss, we use a common combination loss terms for motion forecasting. Following Zhang et al. (2023b), we use the unweighted sum of the negative log-likelihood loss for positions, cross-entropy for confidences, the cosine loss for the heading angle, and the Huber loss for velocities as motion loss. We use AdamW (Loshchilov & Hutter, 2019) in its default configuration as optimizer and set the initial learning rate to 2×10^{-4}. All models have a hidden dimension of 128 and are configured to forecast $k = 6$ motion modes per agent. As post-processing, we do not perform trajectory aggregation, but follow Zhang et al. (2023b) and modify only the predicted confidences of redundant forecasts.

Sparse autoencoders. We train sparse autoencoders with sparse intermediate dimensions of 512, 256, 128, 64, 32, and 16. The total loss combines L2 reconstruction loss with an L1 sparsity penalty: L2 ensures accurate reconstruction, while L1 promotes sparsity by minimizing small, noise-like activations. The L1 must be carefully scaled to avoid deadening important features (Rajamanoharan

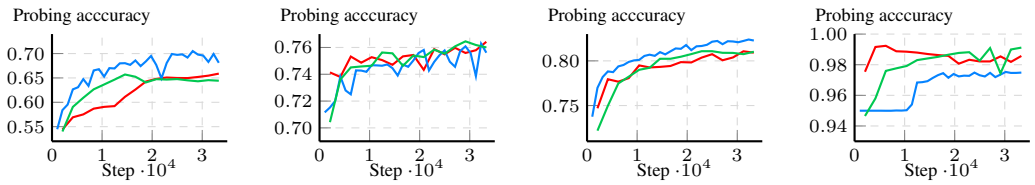
et al., 2024a). We scale it scaled by 3×10^{-4}. We optimize the models over 10 000 epochs using the Adam optimizer (Kingma & Ba, 2015). The final loss values are provided in the appendix.

5 RESULTS

5.1 EXTRACTING INTERPRETABLE FEATURES FOR MOTION

Our approach relies on a well-structured latent space, where samples are clustered with respect to classes of features. Before evaluating the clustering behaviour, we ensure that our features are not highly correlated, as confirmed by the Spearman feature correlation analysis in the appendix. We report linear probing accuracy for interpretable features during training on both datasets.

Figure 2 shows the linear probing accuracies for our interpretable motion features on the Argoverse 2 Forecasting (abbr. 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 a slightly higher acceleration and lower agent accuracies. Overall, high linear probing accuracies for all motion features are achieved. This shows that all models likely exhibit neural collapse towards interpretable motion features.



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

Figure 2: Linear accuracies for RedMotion, Wayformer, and HPTR on the validation split of the AV2F dataset.

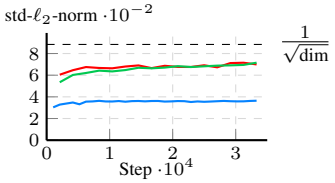
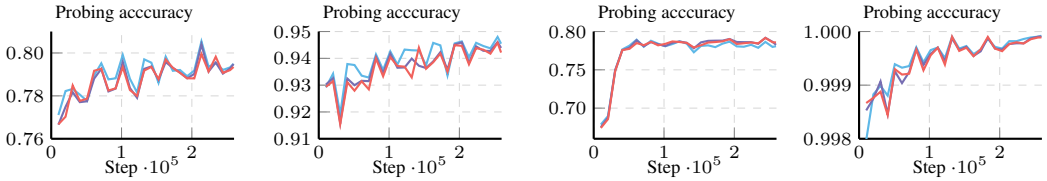


Figure 3: Normalized standard deviation representation quality metric for RedMotion, Wayformer, and HPTR.

The representation quality metric normalized standard deviation of embeddings is shown in Figure 3. Both HPTR and RedMotion learn to generate embeddings with a normalized standard deviation close to the desired value of $1/\sqrt{\text{dim}}$. While the Wayformer model achieves lower scores in this metric. This reflects differences between attention-based and MLP-based motion encoders.

Figure 4 shows the linear probing accuracies for our interpretable motion features on the Waymo dataset. Here, we report the scores for each of the three hidden states H_i in the RedMotion model (i.e., after each module m in the motion encoder, see Figure 1). Similar accuracy scores are reached for all features at all three hidden states. The accuracies for the acceleration and speed features progressively improve, while the direction feature reach a score of 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 s vs. 5 s), which limits the amount possible movements and thus simplifies classifying direction. In contrast to the AV2F dataset, higher accuracies for the speed class are achieved. Overall, the highest scores are reached for the agent features, alike on the AV2F dataset.

On the Waymo dataset, the within-class and between-class normalized variance values for RedMotion are 10.68 and 11.24, respectively, resulting in a class-distance normalized variance (CDNV) of 0.95 (Galanti et al., 2022). On the AV2F dataset, these values are 5.73 and 2.32, yielding a CDNV of 2.46. We hypothesize that the higher CDNV value on AV2F is caused by the longer past motion sequence (i.e., 5 s vs. 1.1 s on Waymo), allowing for a greater range of potential movements.



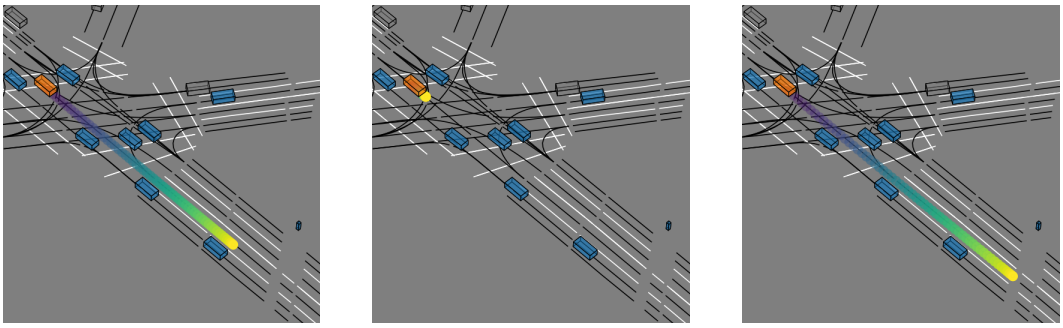
(a) Acceleration (b) Speed (c) Direction (d) Agent

Figure 4: Linear probing accuracies at module 0, module 1 and module 2 for acceleration, direction, and speed on the validation split of the Waymo dataset.

5.2 Qualitative results for controlling at inference

Building on the insight that hidden states are likely collapsed towards our quantized motion features, we fit control vectors to opposing motion features. These control vectors allow for controlling motion forecasts at inference. Specifically, we use our quantized motion features to build pairs of opposing features for the AV2F and the Waymo dataset. Afterwards, we fit sets of control vectors (V_i in Figure 1) as described in Section 3.3.

Figure 5 shows a qualitative example from the AV2F dataset. In subfigure (b) and (c), we apply our acceleration control vector with $\tau = -20$ and $\tau = 100$ to enforce a strong deceleration and a moderate acceleration, respectively.



(a) Default motion forecast (b) Enforced strong deceleration (c) Enforced acceleration

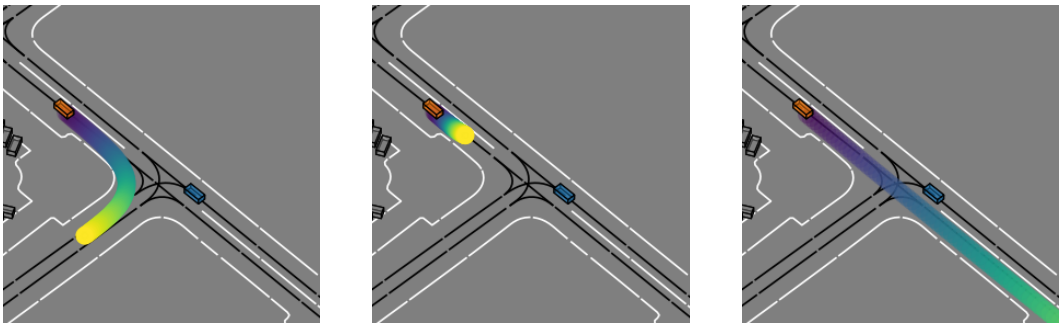
Figure 5: **Controlling a vehicle at an intersection.** In subfigure (b) and (c), we apply our acceleration control vector with $\tau = -20$ and $\tau = 100$ to enforce a strong deceleration and a moderate acceleration. The focal agent is highlighted in orange, dynamic agents are blue, and static agents are grey. Lanes are black lines and road markings are white lines.

Figure 6 shows qualitative results on the Waymo dataset for controlling a motion forecast with the set of control vectors for speed and different temperatures τ. Subfigure (a) shows the default (i.e., non-controlled) top-1 (i.e., most likely) motion forecast. In subfigure (b) and (c), we apply our speed control vector to de- and increase the driven speed of a vehicle. Both controls affect the future speed analogously, while increasing the speed also changes the route to fit the given environment context (i.e., lanes and surrounding vehicles).

In the appendix, we include an example of our direction control and stability under varying temperatures. Overall, these qualitative results support the finding that the hidden states of motion sequences are arranged with respect to our discrete sets of motion features.

5.3 Quantitative comparison of control vectors

In Section 3.3, we emphasized the advantage of sparse autoencoders in generating control vectors that establish a more linear relationship between control vector temperatures and forecasts. In this section, we evaluate how control vectors obtained using sparse autoencoders differ from those derived via plain PCA.



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

Figure 6: **Controlling a vehicle before a predicted right turn.** In subfigure (b) and (c), we apply our speed control vector to de- and increase the driven speed of a vehicle. Both controls affect the future speed analogously, while increasing the speed also changes the route to fit the given environment context (lanes and surrounding vehicles).

For comparison, we train sparse autoencoders (SAE) with varying sparse intermediate dimensions: 512, 256, 128, 64, 32, and 16. For each control vector, we calculate its cosine similarity with control vectors for controlling other features. Table 1 presents the cosine similarity values between control vectors of speed, acceleration, direction, and agent generated with plain PCA and our SAE with a sparse intermediate dimension 128. As expected, the similarity between speed and acceleration, speed and agent, and acceleration and agent is notably high, while the similarity involving direction and other vectors is significantly lower. This result aligns with expectations, as positive speed and acceleration controls lead to faster movement, and our agent control vector represents transition between agent types from pedestrian to vehicle, which is associated with faster movement, as well. Cosine similarity values for the other sparse intermediate dimensions are provided in Table 4 in the appendix. Among these values, the similarity between SAE of intermediate dimension of 128 is the highest.

Table 1: Comparison of control vectors obtained using plain PCA and SAE, measured by cosine similarity (in degrees).

Plain PCA & Plain PCA	speed	acceleration	direction	agent
speed	0.0	11.5	122.6	10.9
acceleration		0.0	126.8	6.8
direction			0.0	128.7
agent				0.0

SAE & SAE	speed	acceleration	direction	agent
speed	0.0	9.5	120.6	7.8
acceleration		0.0	122.9	7.0
direction			0.0	125.8
agent				0.0

We evaluate sparse autoencoders with different activation functions and layer types for a hidden dimension of 128. Following Rajamanoharan et al. (2024b), 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.

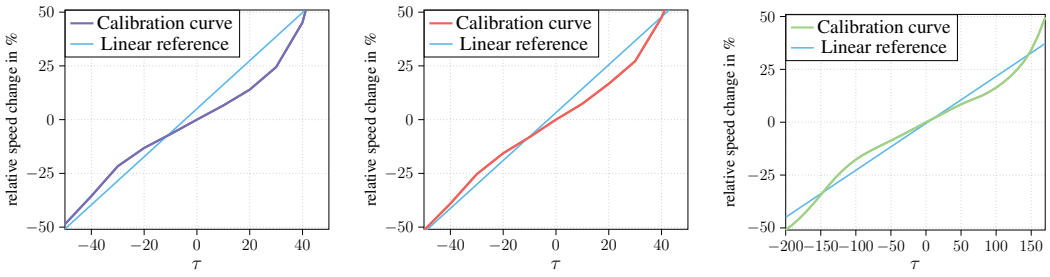
We empirically analyze the temporal causal relationship between hidden states of past motion and motion forecasts. Specifically, we measure the linearity between temperature-scaled activation steering with our speed control vectors and relative speed changes in forecasts. We use the Pearson correlation coefficient, the coefficient of determination (R^2), and the straightness index (S-idx) (Benhamou, 2004) as linearity measures. Given the large range of scenarios in the Waymo dataset, we focus on relative speed changes within a range of $\pm 50\%$ (see Appendix A.16). Higher linearity implies improved controllability.

Table 2 presents linearity measures for activation steering with 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 ConvSAE with a kernel size of 64 achieves the highest straightness index, yet the lowest R^2 scores. As shown in Figure 7 the range of temperatures is much higher

for this ConvSAE than for e.g. the regular SAE. This lowers the R^2 score but does not affect the straightness index. We hypothesize that this is due to feature shrinkage (Rajamanoharan et al., 2024b). Therefore, the JumpReLU configuration of this SAE-type leads to a significantly smaller τ range (see Appendix A.17), which in turn leads to higher R^2 scores (see Table 2). Notably, activation steering with our SAE-based control vector has an almost 1-to-1 relationship 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 sparse autoencoders indicates that sparse intermediate representations capture more distinct features.

Table 2: Linearity measures for activation steering with control vectors: Pearson correlation coefficient, coefficient of determination (R^2), and straightness index.

Autoencoder	Activation function	Pooling	Patch/kernel size	Pearson	R^2	S-idx
–	–	PCA	–	0.988	0.969	0.981
SAE	ReLU	PCA	–	**0.993**	0.984	0.988
SAE	JumpReLU	PCA	–	**0.993**	**0.986**	0.988
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



(a) Plain PCA (b) SAE (c) ConvSAE k64

Figure 7: Calibration curves for activation steering with plain PCA and SAE-based speed control vectors for relative speed changes in forecasts of $\pm 50\%$.

We ablate the sparse intermediate dimensions of regular sparse autoencoders (see Table 7), as well as the sensitivity of our method to hidden states from different modules (see Table 8) and varying speed thresholds (see Table 9). 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.

5.4 Zero-shot generalization with control vectors

Domain shifts between training and test data significantly degrade the performance of many learning algorithms. Zero-shot generalization methods compensate for such domain shifts without further training or fine-tuning (Kodirov et al., 2015; Xian et al., 2017; Mistretta et al., 2024). In motion forecasting, common domain shifts are more or less aggressive driving styles resulting in higher or lower future speeds. 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 upsample this sequence to the original length using linear interpolation.

Table 3 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 the appendix. Without the use of our control vectors, high distance-based errors, miss, and overlap rates are obtained. Using the calibration curve in Figure 7b, 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 as well.

Table 3: Zero-shot generalization to a Waymo dataset version with reduced future speeds using the SAE-128 control vector. Best scores are **bold**, second best are underlined.

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	1.685	4.838	2.870	8.429	0.179	**0.224**
SAE-128	−50	**1.174**	**2.759**	**1.798**	**4.329**	**0.174**	0.236
SAE-128	−70	1.808	3.576	2.035	3.676	0.189	0.302

6 Conclusion

In this work, we take a significant step towards mechanistic interpretability and controllability of motion transformers. We analyze "words in motion" by examining the representations associated with motion features. First, we quantize motion features into discrete subclasses as in natural language. Our experiments on large-scale motion datasets include models with varying environment representations and fusion mechanisms. Furthermore, we analyze the hidden states of attention and MLP-based motion encoders. Specifically, we show that neural collapse towards human-interpretable classes of features occurs in recent motion transformers. Building on these insights, we fit control vectors to opposing features, which allow for controlling forecasts at inference. We further refine this approach by optimizing our control vectors using sparse autoencoding. Notably, this results in a more linear relationship between control vector temperatures and forecasts. This supports the effectiveness of sparse dictionary learning and the use of sparse autoencoders for interpretability. With increased interpretability of our control vectors, we compensate for domain shift and enable zero-shot generalization to unseen dataset characteristics.

Our findings not only improve the practical applicability of recent motion transformer models, but also enable interpreting and manipulating internal representations of transformer models. Possible applications in self-driving vehicles include applying control vectors to motion planning and adjusting planned trajectories, whenever required. Future work can explore using other embedding methods (e.g., Schneider et al. (2023)), as well as features from other modalities by incorporating both static and dynamic scene elements.

References

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In *ICLR*, 2017.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a Visual Language Model for Few-Shot Learning. *NeurIPS*, 2022.

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Linear Algebraic Structure of Word Senses, with Applications to Polysemy. *Transactions of the Association for Computational Linguistics*, 6:483–495, 2018.

Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 2024.

Federico Barbero, Andrea Banino, Steven Kapturowski, Dharshan Kumaran, João GM Araújo, Alex Vitvitskyi, Razvan Pascanu, and Petar Veličković. Transformers need glasses! Information over-squashing in language tasks. *arXiv:2406.04267*, 2024.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The Long-Document Transformer. *arXiv:2004.05150*, 2020.

Ido Ben-Shaul, Ravid Shwartz-Ziv, Tomer Galanti, Shai Dekel, and Yann LeCun. Reverse Engineering Self-Supervised Learning. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *NeurIPS*, 2023.

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

Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your Latents: High-Resolution Video Synthesis with Latent Diffusion Models. In *CVPR*, 2023.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nicholas L Turner, Cem Anil, Amanda Askell, et al. Towards Monosemanticity: Decomposing Language Models With Dictionary Learning. *Transformer Circuits Thread*, 2023.

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

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, et al. Language Models are Few-Shot Learners. *arXiv:2005.14165*, 2020.

Xinlei Chen and Kaiming He. Exploring Simple Siamese Representation Learning. In *CVPR*, 2021.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse Autoencoders Find Highly Interpretable Features in Language Models. *arXiv:2309.08600*, 2023.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, et al. Toy models of superposition. *Transformer Circuits Thread*, 2022.

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 *ICCV*, 2021.

Daocheng Fu, Xin Li, Licheng Wen, Min Dou, Pinlong Cai, Botian Shi, and Yu Qiao. Drive Like a Human: Rethinking Autonomous Driving with Large Language Models. In *WACV*, 2024.

Tomer Galanti, András György, and Marcus Hutter. On the Role of Neural Collapse in Transfer Learning. In *ICLR*, 2022.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging Pretrained Large Language Models to Construct and Utilize World Models for Model-based Task Planning. In *NeurIPS*, 2023.

Tianyu Hua, Wenxiao Wang, Zihui Xue, Sucheng Ren, Yue Wang, and Hang Zhao. On feature decorrelation in self-supervised learning. In *ICCV*, 2021.

Yidong Huang, Jacob Sansom, Ziqiao Ma, Felix Gervits, and Joyce Chai. DriVLMe: Enhancing LLM-based Autonomous Driving Agents with Embodied and Social Experiences. In *First Vision and Language for Autonomous Driving and Robotics Workshop*, 2024.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *ICML*, 2021.

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:2001.08361*, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015.

Elyor Kodirov, Tao Xiang, Zhenyong Fu, and Shaogang Gong. Unsupervised domain adaptation for zero-shot learning. In *ICCV*, 2015.

Yen-Ling Kuo, Xin Huang, Andrei Barbu, Stephen G McGill, Boris Katz, John J Leonard, and Guy Rosman. Trajectory prediction with linguistic representations. In *ICRA*, 2022.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual Instruction Tuning. *NeurIPS*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *ICLR*, 2019.

Jiageng Mao, Junjie Ye, Yuxi Qian, Marco Pavone, and Yue Wang. A Language Agent for Autonomous Driving. *arXiv:2311.10813*, 2023.

Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. In *Proceedings of the North American Association for Computational Linguistics*, 2013.

Marco Mistretta, Alberto Baldrati, Marco Bertini, and Andrew D Bagdanov. Improving zero-shot generalization of learned prompts via unsupervised knowledge distillation. In *ECCV*, 2024.

Nigamaa Nayakanti, Rami Al-Rfou, Aurick Zhou, Kratarth Goel, Khaled S Refaat, and Benjamin Sapp. Wayformer: Motion Forecasting via Simple & Efficient Attention Networks. In *ICRA*, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, et al. Training language models to follow instructions with human feedback. *NeurIPS*, 2022.

Vardan Papyan, X. Y. Han, and David L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *PNAS*, 2020.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

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:2404.10179*, 2024.

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 *ICML*, 2021.

Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Tom Lieberum, Vikrant Varma, Janos Kramar, Rohin Shah, and Neel Nanda. Improving Sparse Decomposition of Language Model Activations with Gated Sparse Autoencoders. In *ICML Workshop on Mechanistic Interpretability*, 2024a.

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 preprint arXiv:2407.14435*, 2024b.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *ICML*, 2021.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. Steering Llama 2 via Contrastive Activation Addition. *arXiv:2312.06681*, 2023.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask Prompted Training Enables Zero-Shot Task Generalization. *arXiv:2110.08207*, 2021.

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 *ICCV*, 2023.

Hao Shao, Yuxuan Hu, Letian Wang, Steven L. Waslander, Yu Liu, and Hongsheng Li. LMDrive: Closed-Loop End-to-End Driving with Large Language Models. In *CVPR*, 2024.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. CLIPort:What and Where Pathways for Robotic Manipulation. In *CoRL*, 2021.

Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Ping Luo, Andreas Geiger, and Hongyang Li. DriveLM: Driving with Graph Visual Question Answering. *arXiv:2312.14150*, 2023.

Nishant Subramani, Nivedita Suresh, and Matthew E Peters. Extracting Latent Steering Vectors from Pretrained Language Models. In *Findings of the Association for Computational Linguistics*, pp. 566–581, 2022.

Shuhan Tan, Boris Ivanovic, Xinshuo Weng, Marco Pavone, and Philipp Kraehenbuehl. Language Conditioned Traffic Generation. In *CoRL*, 2023.

Guy Tevet, Brian Gordon, Amir Hertz, Amit H. Bermano, and Daniel Cohen-Or. MotionCLIP: Exposing Human Motion Generation to CLIP Space. In *ECCV*, 2022.

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 *NeurIPS*, 2021.

Alex Turner, Lisa Thiergart, David Udell, Gavin Leech, Ulisse Mini, and Monte MacDiarmid. Activation Addition: Steering Language Models Without Optimization. *arXiv:2308.10248*, 2023.

Royden Wagner, Ömer Şahin Taş, Marvin Klemp, Carlos Fernandez, and Christoph Stiller. RedMotion: Motion Prediction via Redundancy Reduction. *TMLR*, 2024.

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 *CVPR*, 2023a.

Tsun-Hsuan Wang, Alaa Maalouf, Wei Xiao, Yutong Ban, Alexander Amini, Guy Rosman, Sertac Karaman, and Daniela Rus. Drive Anywhere: Generalizable End-to-end Autonomous Driving with Multi-modal Foundation Models. *arXiv:2310.17642*, 2023b.

Wayve Technologies Ltd. LINGO-1: Exploring Natural Language for Autonomous Driving, 2023. URL `https://wayve.ai/thinking/lingo-natural-language-autonomous-driving/`. Blog Post.

Licheng Wen, Daocheng Fu, Xin Li, Xinyu Cai, Tao MA, Pinlong Cai, Min Dou, Botian Shi, Liang He, and Yu Qiao. DiLu: A Knowledge-Driven Approach to Autonomous Driving with Large Language Models. In *ICLR*, 2024.

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:2301.00493*, 2023.

Menghua Wu, Hao Zhu, Linjia Huang, Yiyu Zhuang, Yuanxun Lu, and Xun Cao. High-fidelity 3D Face Generation from Natural Language Descriptions. In *CVPR*, 2023.

Robert Wu and Vardan Papyan. Linguistic Collapse: Neural Collapse in (Large) Language Models. In *NeurIPS*, 2024.

Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-shot learning-the good, the bad and the ugly. In *CVPR*, 2017.

Zhenhua Xu, Yujia Zhang, Enze Xie, Zhen Zhao, Yong Guo, Kenneth KY Wong, Zhenguo Li, and Hengshuang Zhao. DriveGPT4: Interpretable End-to-end Autonomous Driving via Large Language Modell. *arXiv:2310.01412*, 2023.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding Conditional Control to Text-to-Image Diffusion Models. In *ICCV*, 2023a.

Zhejun Zhang, Alexander Liniger, Christos Sakaridis, Fisher Yu, and Luc Van Gool. Real-Time Motion Prediction via Heterogeneous Polyline Transformer with Relative Pose Encoding. In *NeurIPS*, 2023b.

Xiaoji Zheng, Lixiu Wu, Zhijie Yan, Yuanrong Tang, Hao Zhao, Chen Zhong, Bokui Chen, and Jiangtao Gong. Large Language Models Powered Context-aware Motion Prediction. *arXiv:2403.11057*, 2024.

Ziyuan Zhong, Davis Rempe, Yuxiao Chen, Boris Ivanovic, Yulong Cao, Danfei Xu, Marco Pavone, and Baishakhi Ray. Language-Guided Traffic Simulation via Scene-Level Diffusion. In *CoRL*, 2023.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *CoRL*, 2023.

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.

A Appendix

A.1 Comparison of control vectors using plain PCA and SAE across various sparse intermediate dimensions

Table 4: Comparison of cosine similarity in degrees between control vectors using PCA and SAE across various sparse intermediate dimensions (512, 256, 128, 64, 32, 16). The tables on the left represent the cosine similarity between control vectors of the same SAE model, whereas those on the right represent the cosine similarity between control vectors of SAE and those derived from Plain PCA. The control vector with a sparse intermediate dimension of 128 achieves the highest overall similarity.

SAE-512 & SAE-512	speed	acceleration	direction	agent
speed	0.0	10.2	121.8	7.6
acceleration		0.0	123.7	7.6
direction			0.0	126.9
agent				0.0

Plain PCA & SAE-512	speed	acceleration	direction	agent
speed	20.7	28.6	123.8	23.4
acceleration	19.1	23.0	128.5	18.6
direction	115.9	116.6	13.7	120.8
agent	19.4	24.4	130.2	18.3

SAE-256 & SAE-256	speed	acceleration	direction	agent
speed	0.0	9.9	120.9	7.9
acceleration		0.0	123.7	7.2
direction			0.0	126.3
agent				0.0

Plain PCA & SAE-256	speed	acceleration	direction	agent
speed	21.5	26.8	123.8	23.3
acceleration	20.3	21.0	128.7	18.7
direction	114.7	116.9	13.7	120.1
agent	20.8	23.1	130.2	18.7

SAE-128 & SAE-128	speed	acceleration	direction	agent
speed	0.0	9.5	120.6	7.8
acceleration		0.0	122.9	7.0
direction			0.0	125.8
agent				0.0

Plain PCA & SAE-128	speed	acceleration	direction	agent
speed	19.7	25.3	124.3	21.6
acceleration	19.2	20.0	128.8	17.5
direction	115.2	117.1	12.1	120.5
agent	19.5	21.8	130.4	17.1

SAE-64 & SAE-64	speed	acceleration	direction	agent
speed	0.0	9.7	121.0	8.0
acceleration		0.0	123.2	7.5
direction			0.0	126.3
agent				0.0

Plain PCA & SAE-64	speed	acceleration	direction	agent
speed	18.1	23.7	124.7	19.3
acceleration	19.3	19.9	128.9	16.5
direction	115.0	116.6	13.3	120.5
agent	19.8	21.9	130.5	16.4

SAE-32 & SAE-32	speed	acceleration	direction	agent
speed	0.0	9.8	120.3	8.3
acceleration		0.0	122.8	7.0
direction			0.0	125.8
agent				0.0

Plain PCA & SAE-32	speed	acceleration	direction	agent
speed	14.7	18.8	126.4	15.5
acceleration	18.0	15.5	130.3	14.1
direction	114.4	116.9	10.9	120.2
agent	18.1	17.6	132.0	13.4

SAE-16 & SAE-16	speed	acceleration	direction	agent
speed	0.0	9.5	124.1	9.3
acceleration		0.0	125.2	7.5
direction			0.0	129.3
agent				0.0

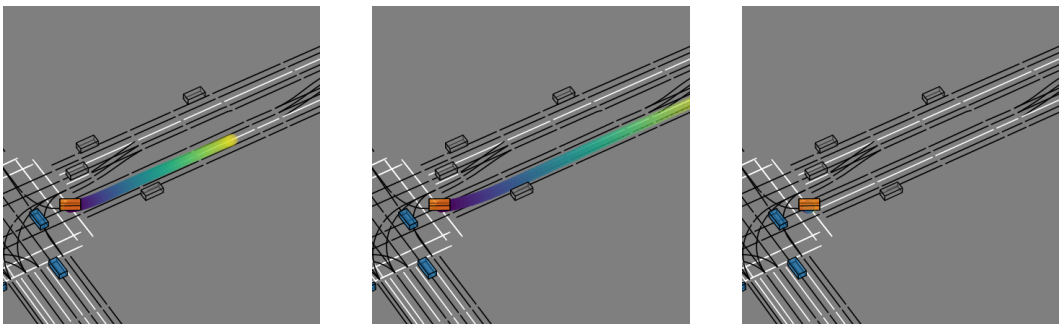
Plain PCA & SAE-16	speed	acceleration	direction	agent
speed	23.5	25.1	126.6	21.8
acceleration	28.4	26.0	128.9	23.5
direction	110.2	111.9	24.6	116.6
agent	28.0	26.8	131.0	22.5

Table 5: Loss metrics for SAEs across sparse intermediate dimensions, trained for 10.000 epochs.

Dim	#Epochs	Total loss	L1 loss	L2 loss	Total reconst loss
512	9805	4.01	1.52	8270.70	0.0016
256	9845	3.72	1.38	7823.98	0.0014
128	9820	4.14	1.56	8608.95	0.0017
64	9348	4.56	1.89	8894.97	0.0019
32	9864	7.14	3.90	10 795.54	0.0043
16	9956	17.44	13.37	13 576.57	0.0142

A.2 Additional qualitative results

Figure 8 shows a qualitative example for our direction control 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 8: **Controlling a left turning vehicle.** In subfigure (b) and (c), we apply our left-direction control vector and right-direction control vector. The focal agent is highlighted in orange, dynamic agents are blue, and static agents are grey. Lanes are black lines and road markings are white lines.

A.3 Natural language as a modality in robotics and self-driving

We provide examples illustrating how natural language is utilized as a modality in robotics and autonomous driving, building upon the general approaches introduced in the main body of the text.

Conditioning. Tan et al. (2023); Zhong et al. (2023) generate dynamic traffic scenes based on user-specified descriptions expressed in natural language.

Prompting. Kuo et al. (2022) generate linguistic descriptions of predicted trajectories during decoding, capturing essential information about future maneuvers and interactions. More recent works employ large language models (LLMs) to analyze driving environments in a human-like manner, providing explanations of driving actions and the underlying reasoning (Xu et al., 2023; Fu et al., 2024; Sima et al., 2023; Wayve Technologies Ltd., 2023). This offers a human-centric description of the driving environment and the model's decision-making capabilities.

Enriching. Zheng et al. (2024) integrate the enriched context information of LLMs into motion forecasting models. Wang et al. (2023b) use LLMs for data augmentation to improve out-of-distribution generalization. Others use pre-trained LLMs for better generalization during decision-making (Mao et al., 2023; Wen et al., 2024; Shao et al., 2024).

Instructing. Shridhar et al. (2021) enable robotic control through language-based instruction. Zitkovich et al. (2023) incorporate web knowledge, enriching vision-language-action models for more generalized task performance. Huang et al. (2024) demonstrate the use of instructions to guide task execution in self-driving, with experiments in simulation environments.

Although these works create a language interfaces to interact with the underlying model, in contrast to our work, they do not measure the degree to which human-interpretable motion features are embedded within hidden states.

A.4 Parameters for classifying motion features

We classify motion trajectories with a sum less than $15°$ degrees 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 $25\,\mathrm{km\,h^{-1}}$ and $50\,\mathrm{km\,h^{-1}}$ 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.

Figure 9 presents the distribution of motion subclasses across the datasets. 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 (65%) and traveling straight (49%), while the AV2F dataset showcases a higher proportion of accelerating instances (52%). Additionally, AV2F stands out with 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 potentially focusing on highway or structured urban driving, while AV2F encompasses more diverse traffic situations.

A.5 Early, hierarchical and late fusion in motion encoders

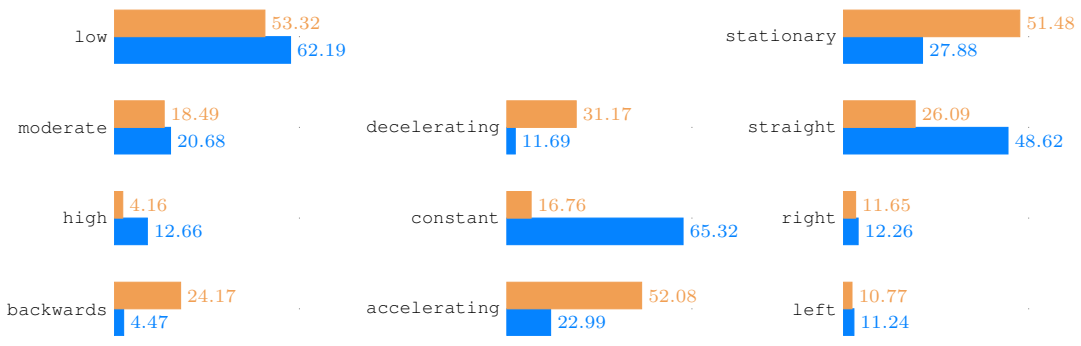
We define fusion types for motion transformers based on the information processed in the first attention layers within a model. In early fusion, the first attention layers process motion data of the modeled agent, other agents, and environment context. In hierarchical fusion, the first attention layers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

A.6 Motion forecasting metrics

Following (Wilson et al., 2023; Ettinger et al., 2021), we use the average displacement error (minADE), the final displacement error (minFDE), and their respective Brier variants, which account for the predicted confidences. Furthermore, we compute the miss rate, and overlap rate to evaluate motion forecasts. All metrics are computed using the minimum mode for $k = 6$ modes. Accordingly, the metrics for the mode closest to the ground truth are measured.

A.7 Title origin

The title of our work "words in motion" is inspired by our quantization method using natural language and a common notion in the computer architecture literature. In computer architecture, a word is a natural unit of data for a processing unit (e.g., CPU or GPU). In our work, words are classes of motion features, which are embedded in the hidden states of motion sequences processed by motion forecasting models.



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

Figure 9: Distributions of our motion features for the Argoverse 2 Forecasting and the Waymo Open Motion datasets. All numbers are percentages.

A.8 FEATURE CORRELATION

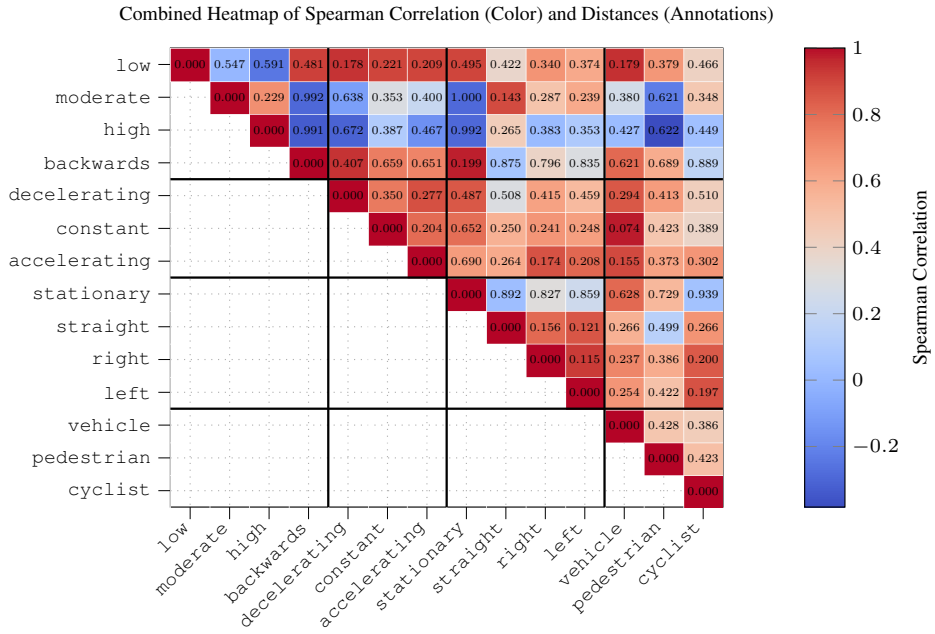


Figure 10: Heatmap representing Spearman correlation between feature cluster means for the Waymo Open Motion dataset. The values in the matrix indicate pairwise distances between clusters, normalized by the largest distance.

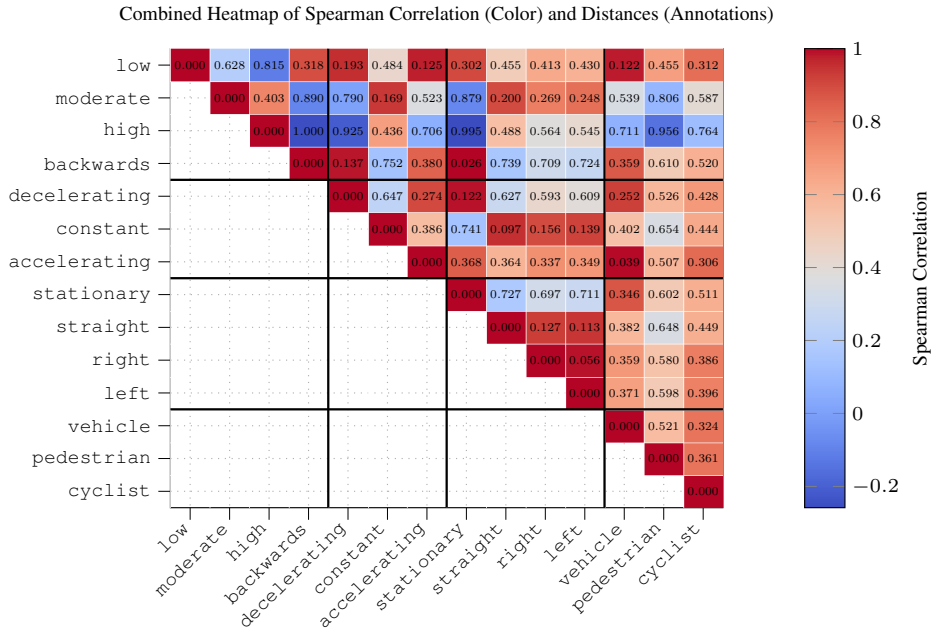


Figure 11: Heatmap representing Spearman correlation between feature cluster means for the Argoverse 2 Forecasting dataset. The values in the matrix indicate pairwise distances between clusters, normalized by the largest distance.

A.9 EXPLAINED VARIANCE

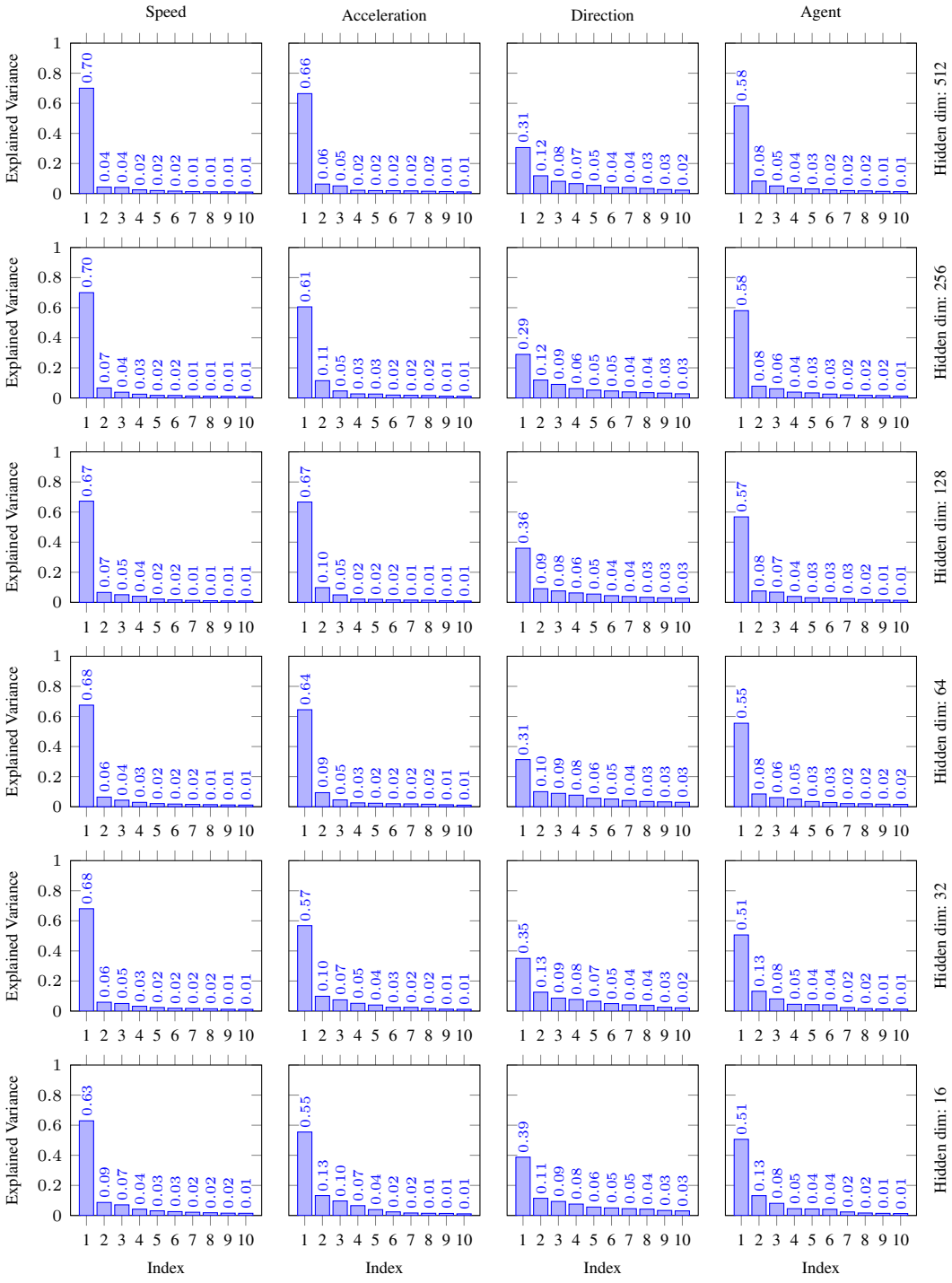


Figure 12: Explained variance for SAE across hidden latent dimensions 512, 256, 128, 64, 32, and 16 shown vertically.

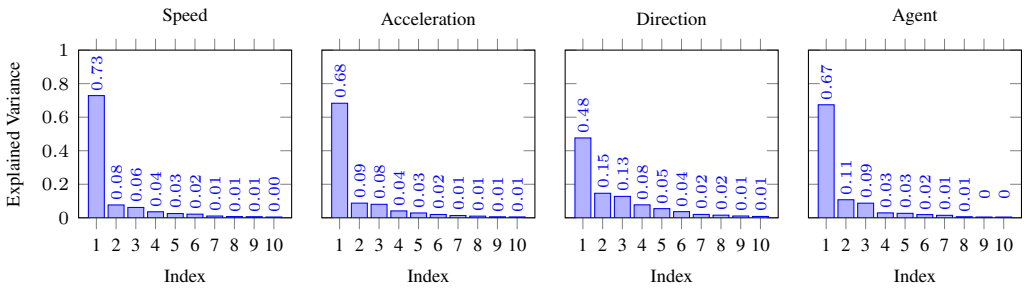


Figure 13: Explained variance for Plain-PCA.

A.10 Neural collapse

Neural collapse metrics capture structural patterns in feature representations, focusing on clustering, geometry, and alignment. Class Distance Normalized Variance (CDNV), also referred to as "$\mathcal{NC}1$", 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. CDNV provides a robust alternative to methods that compare between- and within-cluster variation for assessing feature separability (Galanti et al., 2022).

$$\mathcal{NC}1^{\text{cdnv}}_{c,c'} = \frac{\sigma_c^2 + \sigma_{c'}^2}{2\|\mu_c - \mu_c'\|_2^2}, \quad \forall c \neq c'$$

A.11 Inference latency

Table 6 shows inference latency measurements of a RedMotion model on the Waymo Open Motion dataset with and without activation steering with our control vectors. Our activation steering adds only about 1ms to the total inference latency. Since most datasets are recorded at 10Hz (e.g., Wilson et al. (2023); Ettinger et al. (2021)), it is common to define the threshold for real-time capability of self-driving stacks as $\leq$100ms. Considering the inference latency of recent 3D perception models (e.g., approx. 40ms for Wang et al. (2023a)), which must be called before motion forecasting, activation steering should not add significantly to the forecasting latency.

Table 6: **Inference latency without and with activation steering 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.

Activation steering	Focal agents	Inference latency
False	8	50.21 ms
True	8	51.08 ms

A.12 Meta-Architecture of multimodal motion transformers

We study multimodal motion transformers (Nayakanti et al., 2023; Wagner et al., 2024; Zhang et al., 2023b), which process motion, lane and traffic light data. The meta-architecture of these models is shown in Figure 14. These models generate motion M_i, map K_j, and traffic light T_k tokens using learned tokenizers. Modality-specific encoders aggregate information of multiple tokens with attention mechanisms (e.g., across multiple past timesteps for motion tokens). Afterwards, learned motion queries Q (i.e., a form of learned anchors) cross-attend in the motion decoder to M, K, and T. Finally, a learned de-tokenizer projects the last hidden state of Q into multiple motion forecasts, which are represented as 2D Gaussians for future positions in bird's-eye-view and associated confidences. The difference between the models lay in the type of attention and fusion mechanisms as well as the used reference frames.

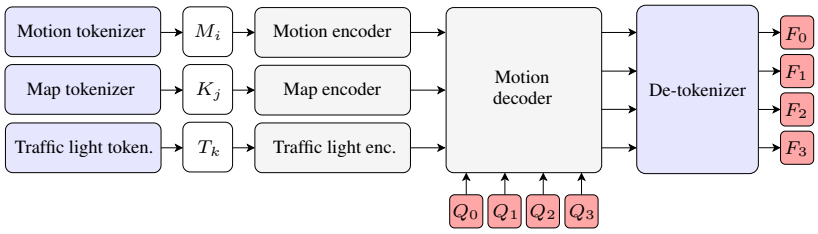


Figure 14: Motion transformer meta architecture of RedMotion, Wayformer, and HPTR.

A.13 Linearity analysis across sparse autoencoder dimensions

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

Table 7: **Scaling sparse autoencoders.** SAE-128 has a sparse intermediate dimension of 128 and achieves the highest linearity scores for activation steering with our control vector for speed. Linearity measures for controlling: Pearson correlation coefficient, coefficient of determination (R^2), and straightness index.

A.14 Control vectors across modules in sparse autoencoders

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 8: **Generating control vectors for hidden states of different modules.** Control vectors for speed generated in earlier modules achieve lower linearity scores for activation steering. Linearity measures for controlling: Pearson correlation coefficient, coefficient of determination (R^2), and straightness index.

A.15 Sensitivity analysis for various speed thresholds

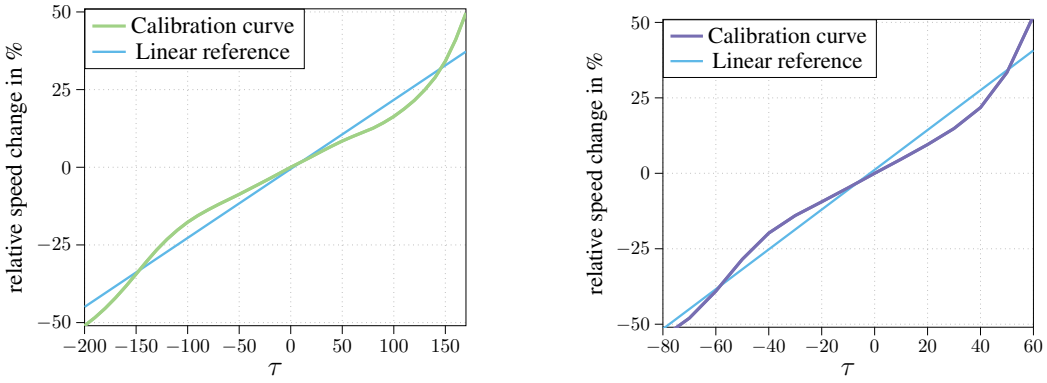
Autoencoder	Low speed	High speed	Pearson	R^2	S-idx
SAE-128	$< 25\,\mathrm{km\,h^{-1}}$	$> 50\,\mathrm{km\,h^{-1}}$	0.993	0.984	0.988
SAE-128	$< 25\,\mathrm{km\,h^{-1}}$	25 to $50\,\mathrm{km\,h^{-1}}$	0.994	0.987	0.989
SAE-128	25 to $50\,\mathrm{km\,h^{-1}}$	$> 50\,\mathrm{km\,h^{-1}}$	0.355	−0.734	0.533

Table 9: **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. Linearity measures for controlling: Pearson correlation coefficient, coefficient of determination (R^2), and straightness index.

A.16 CHOOSING A RANGE OF RELATIVE CHANGES IN FUTURE SPEED

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: $\mu \approx 12\,\mathrm{m\,s^{-1}}$, $\sigma \approx 5\,\mathrm{m\,s^{-1}}$, pedestrians: $\mu \approx 1.5\,\mathrm{m\,s^{-1}}$, $\sigma \approx 0.7\,\mathrm{m\,s^{-1}}$, and cyclists: $\mu \approx 7\,\mathrm{m\,s^{-1}}$, $\sigma \approx 3\,\mathrm{m\,s^{-1}}$) the $\pm 50\%$ range corresponds to speeds within approximately $\pm 1\sigma$ of the mean for each agent type.

A.17 JUMPRELU COMPENSATES FEATURE SHRINKAGE IN CONVSAES



(a) ConvSAE k64

(b) ConvSAE k64 JumpReLU

Figure 15: JumpReLU compensates feature shrinkage as reflected in a smaller range of τ values for the same range of relative speed changes.

The range of temperatures is much higher for the ConvSAE than for the JumpReLU version of this sparse autoencoder (ConvSAE k64 JumpReLU). We hypothesize that this is due to feature shrinkage (Rajamanoharan et al., 2024b). Therefore, the JumpReLU configuration of this SAE-type leads to a significantly smaller τ range, which in turn leads to higher R^2 scores (see Table 2).