Unveiling and Manipulating Concepts in Time Series Foundation Models

Michał Wiliński Mononito Goswami Nina Żukowska Willa Potosnak Artur Dubrawski
Auton Lab, School of Computer Science, Carnegie Mellon University
Pittsburgh, PA 15213
`{mwilinsk,mgoswami}@andrew.cmu.edu`

Abstract

Time series foundation models promise to be powerful tools for a wide range of applications. However, little is known about the concepts that these models learn and how we can manipulate them in the latent space. Our study bridges these gaps by **identifying** concepts learned by these models, **localizing** them to specific parts of the model, and **steering** model predictions along these conceptual directions, using synthetic time series data. Our results show that `MOMENT`, a state-of-the-art foundation model, can discern distinct time series patterns, and that this ability peaks in the middle layers of the network. Moreover, we show that model outputs can be steered using insights from its activations (e.g., by introducing periodic trends to initially constant signals through intervention during inference). Our findings underscore the importance of synthetic data in studying and steering time series foundation models and intervening throughout the whole model (using *steering matrices*), instead of a single layer.

1 Introduction

Foundation models have taken significant strides in modeling both textual [Brown et al., 2020] and visual [Dosovitskiy et al., 2020] data, and have made complex language and image processing accessible to non-experts. These models are pre-trained on massive internet-scale datasets and can be used to solve multiple tasks across a variety of domains, with little to no adaptation. Recently, a growing body of work [Garza and Mergenthaler-Canseco, 2023, Goswami et al., 2024, Rasul et al., 2024, Das et al., 2024, Woo et al., 2024, Ansari et al., 2024] has extended the benefits of this paradigm to time series data, a modality prevalent in fields such as finance [Taylor, 2008], healthcare [Goswami et al., 2021], and climate science [Schneider and Dickinson, 1974].

Time series foundation models (TSFMs) have shown promising performance on multiple modeling tasks such as forecasting, classification, anomaly detection, and imputation, across a wide range of domains, and in settings with varying amounts of data and supervision. However, little is known about *why these models work, the kinds of concepts that these models are learning, and how these concepts can be manipulated to influence model outputs.*

In this paper, we take initial steps towards addressing these gaps by *identifying* concepts learned by these models, *localizing* them to a small subset of their hidden states, and *steering* model predictions along these conceptual directions, using only synthetic time series. **To the best of our knowledge, our work is among the first to identify, localize, and manipulate learned concepts in time series foundation models by solely using carefully generated synthetic time series.** We also demonstrate the effectiveness of intervening throughout the model using *steering matrices* for precise control, in contrast to the single-layer steering vector approach (Fig. 4).

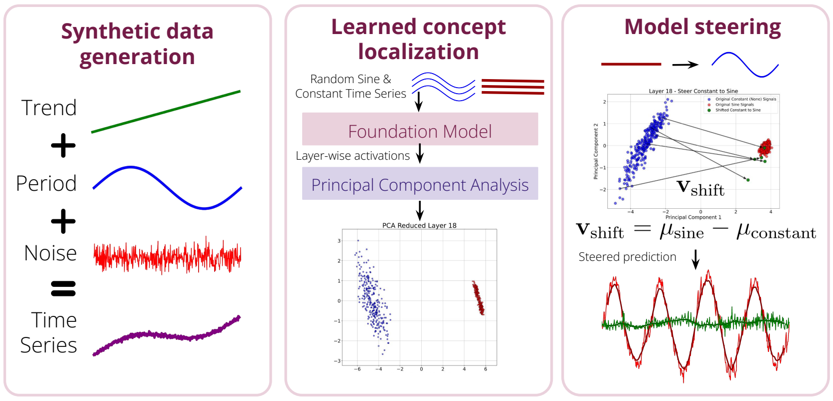

Synthetic data generation
Trend
+
Period
+
Noise
=
Time Series
Learned concept localization
Random Sine & Constant Time Series
Foundation Model
Layer-wise activations
Principal Component Analysis
PCA Reduced Layer 18
Model steering
Layer 18 - Steer Constant to Sine
$\mathbf{v}_{\text{shift}}$
$\mathbf{v}_{\text{shift}} = \mu_{\text{sine}} - \mu_{\text{constant}}$
Steered prediction

Figure 1: Overview of the methodology for synthetic data generation and feature localization in TSFMs. The synthetic data is generated by combining trend, pattern, and noise components. Linear feature localization is performed using principal component analysis and linear models, while interventions are applied to control features in the latent space.

2 Methodology

For brevity, we defer a detailed discussion of related work to Appendix A.

Synthetic Data Generation To investigate the representations learned in TSFMs, we generate synthetic univariate time series by combining three key components: trend, pattern, and noise. This approach provides control over diversity and features included in inputs for our analysis. A typical univariate time series used in our setup $\{x_t\}_{t=1}^{T}$ is defined as:

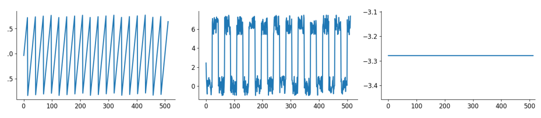


Figure 2: Time series generated using Eq. 1.

$$x_t = \text{Trend}(t) + \text{Period}(t) + \text{Noise}(t), \quad t = 1, 2, \ldots, T\,. \tag{1}$$

Each component serves a specific role: trend models long-term monotonic increase or decrease in time series values, periodic pattern captures periodic or cyclical behaviors, and noise accounts for random fluctuations. This flexible framework enables us to systematically vary input properties and evaluate how different features are represented across model layers. Further information about the data-generating process, mathematical definitions of the components, and hyper-parameters can be found in Appendix B.

Identifying and Localizing Linearly Represented Features. To investigate which features are linearly represented in the model's latent space, we build on the investigation approach outlined in [Marks and Tegmark, 2023]. In particular, we first train a set of linear probes (linear models trained on the model's intermediate activations) to identify the layers with the greatest linear separation of a given feature and then visualize the embeddings in the lower-dimensional space using PCA to directly analyze the representations.[1]

Let $\mathbf{h}^{l}_{\mathbf{M}}(x) \in \mathbb{R}^{s \times d}$ denote the hidden representation of a time series x at layer $l_{\mathbf{M}}$ of a time series foundation model $\mathbf{M}$, where s is the number of symbols (patches in our case), d is the dimensionality of the hidden layer. Our objective is two-fold: (1) to ascertain whether a feature of interest f, such as discriminating a sinusoidal vs. constant pattern (Fig. 2), is represented as a *direction* in $\mathbf{M}$'s latent space. We say that such a feature is linearly represented in $\mathbf{M}$. If f is linearly represented by $\mathbf{M}$, we also want to (2) identify which layer l in $\mathbf{M}$ learns this concept in the most discriminant way.

To determine whether a feature f (e.g., sinusoidal vs. constant time series) is linearly represented, we first: **(1) generate a synthetic time series dataset by varying f.** In our example, the dataset comprises multiple sinusoids and constant time series randomly sampled using Eq. 1. Using this

[1]Note that nonlinear alternatives to PCA can be used to harness non-linearly separable feature artifacts.

dataset, we **(2) extract intermediate representations** $\mathbf{h}^l_{\mathbf{M}}(x)$ of each time series from the residual stream of each layer $l_{\mathbf{M}}$. Finally for each layer, we **(3) find and assess the dominant direction** of all time series in our dataset using linear probes and PCA. We begin by training the probes on the data to discriminate a specific feature, then we assess which of the layers yield the greatest margin of separation, and based on that we perform a closer investigation of the representations of these layers using PCA projection to visualize embeddings in the low-dimensional (2D) space for ease of inspection. This enables an intuitive way to look into the results of the linear probing.

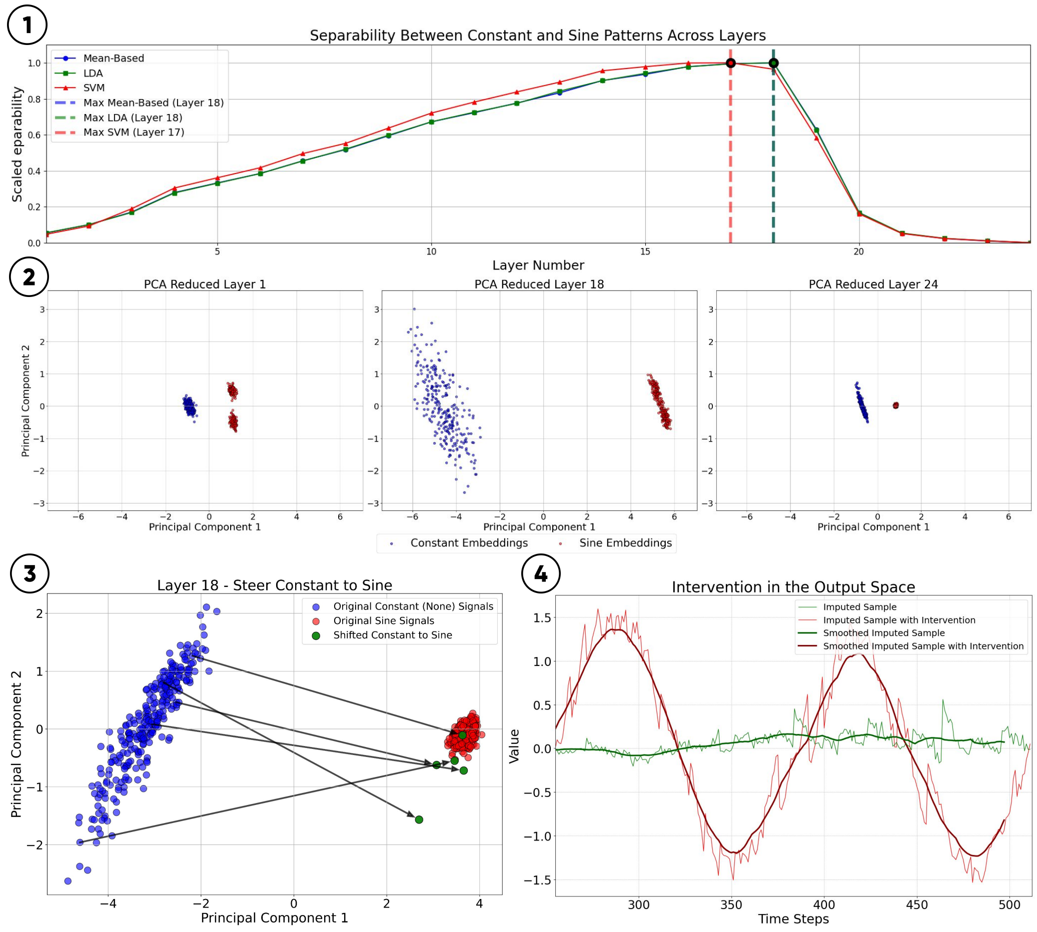


Figure 3: ① Linear separability between constant and sine signals increases through the model, peaking at intermediate layers. ② The 2D projection shows clear separation between the two signal types. ③ Steering demonstrates expected transformations at the representation level. ④ These transformations are effectively reflected in the output space.

Deriving Steering Matrices for Model Steering. After identifying that a feature f is linearly represented in model $\mathbf{M}$'s latent space, we apply interventions by adjusting the hidden representations $\mathbf{h}^l_{\mathbf{M}}(x)$ across multiple layers. Instead of using a single steering vector, we utilize a **steering matrix** $\mathbf{S} \in \mathbb{R}^{L \times s \times d}$, where L is the number of layers, s is the number of symbols (patches), and d is the dimensionality of hidden representations. This matrix allows us to simultaneously intervene across multiple layers, which we found to be more effective than single-layer interventions (Fig. 4).

Matrix $\mathbf{S}$ is calculated by computing the difference between the medoids of hidden representations of time series corresponding to different values of feature f. If $\mathbf{m}^l_{f=\texttt{constant}}$ and $\mathbf{m}^l_{f=\texttt{sine}}$ denote the medoids of constant and time series at layer l, then the steering vector at layer l is given by $\mathbf{S}^l = \mathbf{m}^l_{f=\texttt{sine}} - \mathbf{m}^l_{f=\texttt{constant}}$. We stack these vectors for all layers to derive the steering matrix. During inference, to steer model prediction, at each layer l we update the hidden representation $\mathbf{h}^l_{\mathbf{M}}(x)$ as $\mathbf{h}^l_{\mathbf{M}}(x) \leftarrow \mathbf{h}^l_{\mathbf{M}}(x) + \alpha\mathbf{S}^l$, where α is scalar that controls the strength of the intervention.

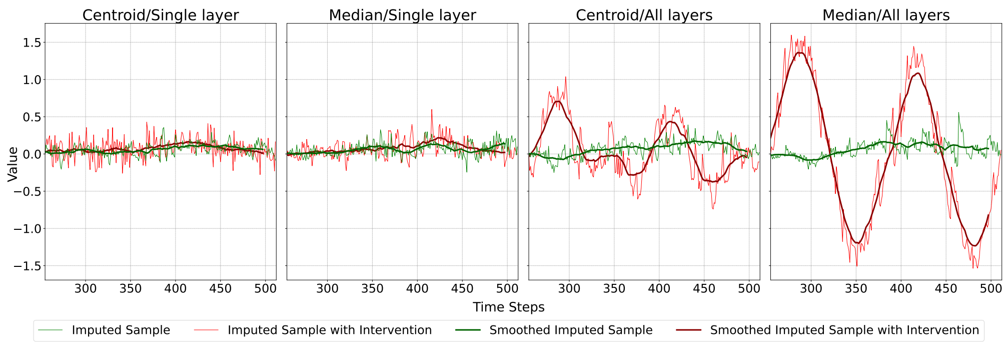


Figure 4: Comparison of 4 different strategies in steering `MOMENT`'s reconstruction outputs, give a constant time series green as input. We found that steering `MOMENT` predictions towards the medoid of the sine signals throughout the model (*rightmost*), was more effective than alternatives such as intervening in a single layer (*leftmost*).

3 Experimental Setup

Research Questions. Our study is structured around the following research questions: **(1) Identify:** Are time series foundation models capable of learning different time series concepts? **(2) Localize:** Which parts of the model learn the concept? **(3) Manipulate:** Can we leverage these learned concepts to bias model predictions?

Experimental Setup. For all our experiments, we utilize the open-source `MOMENT-Large`[2] model introduced by Goswami et al. [2024]. `MOMENT` is pre-trained to reconstruct input time series. We randomly generate a total of $N = 1024$ time series with two basic patterns: constant and sinusoidal (Fig. 2). We use these time series and the methods outlined in the previous section to find out: (1) if `MOMENT` can distinguish constant time series from sinusoidal signals, (2) which layer(s) learn this concept, and (3) how we can introduce periodic (sinusoidal) trends to constant signals by steering `MOMENT`'s predictions (Fig. 3.4).

4 Results and Future Work

Results. Our results are summarized in Fig. 3. The first two parts of the figure clearly show that `MOMENT` can discern constant signals from sinusoidal ones. This capability peaks at layer 18 of the model, as measured by the separation between clusters representing embeddings of constant (blue) and sine (red) signals. Parts 3 and 4 of Fig. 3 illustrate the impact of steering constant signals towards sinusoidal signals, both in the latent space (as shown, in layer 18) and the output space.

Fig. 4 compares four different strategies of steering `MOMENT`'s reconstruction outputs, given a constant time series green as input. We found that steering predictions towards the medoid of the sine signals throughout the model was more effective than alternatives such as mean probing a single layer (Centroid/Single). In Appendix D, we further show that our proposed interventions generalize beyond the time series used to derive the steering matrix, and specifically to time series with varying trends.

Future Work. This paper provides insights into how a few simple patterns are linearly represented in time series foundation models. Future work must evaluate whether time series foundation models can learn more complex patterns present in real-world time series and whether steering matrices estimated using synthetic data can be used to steer predictions of out-of-distribution, real-world time series. While our methods are broadly applicable to different transformer-based foundation models, future research should explore other architectures such as state space models [Gu and Dao, 2023] and stacked multi-layer perceptrons [Ekambaram et al., 2024]. Moreover, future studies should evaluate whether our findings hold for other time series foundation models and tasks such as forecasting, anomaly detection, and classification. Beyond time series, we hope that our work inspires the use of synthetic data to steer large language and vision models as well.

[2]`https://huggingface.co/AutonLab/MOMENT-1-large`

Acknowledgment

This work was partially supported by NSF (awards 2406231 and 2427948), NIH (awards R01NS124642, R01DK131586, and R01NR013912), and the US Army (W911NF-20-D0002).

References

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Syndar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*, 2024.

Amos Azaria and Tom Mitchell. The internal state of an LLM knows when it's lying. *arXiv preprint arXiv:2304.13734*, 2023.

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*, 33:1877–1901, 2020.

Collin Burns, Haotian Ye, Dan Klein, and Jacob Steinhardt. Discovering latent knowledge in language models without supervision. In *The Eleventh International Conference on Learning Representations*, 2023.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, Anthony Bau, and James Glass. What is one grain of sand in the desert? analyzing individual neurons in deep NLP models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6309–6317, 2019.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Forty-first International Conference on Machine Learning*, 2024.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

Vijay Ekambaram, Arindam Jati, Nam H Nguyen, Pankaj Dayama, Chandra Reddy, Wesley M Gifford, and Jayant Kalagnanam. TTMs: Fast multi-level tiny time mixers for improved zero-shot and few-shot forecasting of multivariate time series. *arXiv preprint arXiv:2401.03955*, 2024.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Azul Garza and Max Mergenthaler-Canseco. TimeGPT-1, 2023.

Gabriel Goh, Nick Cammarata, Chelsea Voss, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah. Multimodal neurons in artificial neural networks. *Distill*, 6(3):e30, 2021.

Mononito Goswami, Benedikt Boecking, and Artur Dubrawski. Weak supervision for affordable modeling of electrocardiogram data. In *AMIA Annual Symposium Proceedings*, volume 2021, page 536. American Medical Informatics Association, 2021.

Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. MOMENT: A family of open time-series foundation models. In *International Conference on Machine Learning*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing. *arXiv preprint arXiv:2305.01610*, 2023.

Yong Liu, Haoran Zhang, Chenyu Li, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer: Generative pre-trained transformers are large time series models. In *Forty-first International Conference on Machine Learning*, 2023.

Samuel Marks and Max Tegmark. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets. *arXiv preprint arXiv:2310.06824*, 2023.

Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Hena Ghonia, Rishika Bhagwatkar, Arian Khorasani, Mohammad Javad Darvishi Bayazi, George Adamopoulos, Roland Riachi, Nadhir Hassen, Marin Biloš, Sahil Garg, Anderson Schneider, Nicolas Chapados, Alexandre Drouin, Valentina Zantedeschi, Yuriy Nevmyvaka, and Irina Rish. Lag-Llama: Towards foundation models for probabilistic time series forecasting, 2024.

Stephen H Schneider and Robert E Dickinson. Climate modeling. *Reviews of Geophysics*, 12(3): 447–493, 1974.

Stephen J Taylor. *Modelling Financial Time Series*. World Scientific, 2008.

Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. *arXiv preprint arXiv:2202.07125*, 2022.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. In *Forty-first International Conference on Machine Learning*, 2024.

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 preprint arXiv:2310.01405*, 2023.

A Appendix: Related Work

Time Series Foundation Models. Transformers have long been used for various time series modeling tasks [Wen et al., 2022], but large pre-trained transformers are a recent phenomenon. Garza and Mergenthaler-Canseco [2023] first introduced `TimeGPT`, an encoder-decoder model pre-trained on internet-scale time series data, with encouraging zero-shot forecasting performance. While `TimeGPT` is accessible via API, its weights and datasets remain private. Following this, several open-source pre-trained models have emerged, with slight modifications to the transformer architecture, different pre-training datasets and strategies, primarily focusing on forecasting task [Rasul et al., 2024, Das et al., 2024, Liu et al., 2023, Woo et al., 2024, Ansari et al., 2024]. Concurrently, Goswami et al. [2024] released `MOMENT`, an open family of encoder-only foundation models designed to handle multiple time series tasks beyond forecasting, including classification and anomaly detection. In this paper, we perform experiments on the `MOMENT-Large` backbone, but our methods are broadly applicable to other open-source foundation models as well.

Investigating Representations of Pre-trained Models. Understanding the internal representations learned by pre-trained models has been an active area of research, particularly in the context of LLMs and vision models. Previous studies have explored whether individual neurons or directions in a model's latent space correspond to specific features or concepts [Dalvi et al., 2019, Goh et al., 2021, Gurnee et al., 2023, Elhage et al., 2022]. These investigations often focus on identifying linear representations, where features are encoded as linear combinations of neuron activations. Recent work has also employed various probing techniques to classify and interpret these internal representations, addressing aspects such as truthfulness and model robustness [Azaria and Mitchell, 2023, Zou et al., 2023, Burns et al., 2023, Marks and Tegmark, 2023]. While many of these studies rely on meticulously curated datasets to probe language and vision models, we demonstrate that for time series models, synthetic data generated using simple mechanisms can effectively identify, localize, and probe the linear concepts learned by these models.

B Appendix: Synthetic Data Generation Procedure

The generated time series $\{x_t\}_{t=1}^{T}$ is modeled as a combination of three key components: trend, pattern, and noise. Formally, the time series at time t is given by:

$$x_t = \text{Trend}(t) + \text{Pattern}(t) + \text{Noise}(t), \quad t = 1, 2, \ldots, T$$

Where each component is defined as follows:

B.1 Trend Component

The trend component models the long-term progression of the time series. There are multiple options for generating the trend:

Linear Trend: The linear trend is modeled as:

$$\text{Trend}(t) = \alpha t + \beta$$

Where α is the slope and β is the intercept. These parameters are sampled from uniform distributions:

$$\alpha \sim U(\text{slope}_{\text{min}}, \text{slope}_{\text{max}}), \quad \beta \sim U(\text{intercept}_{\text{min}}, \text{intercept}_{\text{max}})$$

Exponential Trend: The exponential trend is modeled as:

$$\text{Trend}(t) = e^{\gamma t}$$

Where γ is the growth rate sampled from a uniform distribution:

$$\gamma \sim U(\text{growth\_rate}_{\text{min}}, \text{growth\_rate}_{\text{max}})$$

B.2 Pattern Component

The pattern component captures periodic variations. Several options exist for modeling the pattern:

Sine Pattern: The sine pattern is modeled as:

$$\text{Pattern}(t) = A \sin\left(\frac{2\pi t}{P}\right)$$

Where A is the amplitude and P is the period, both sampled from uniform distributions:

$$A \sim U(\text{amplitude}_{\text{min}}, \text{amplitude}_{\text{max}}), \quad P \sim U(\text{period}_{\text{min}}, \text{period}_{\text{max}})$$

Square Pattern: The square wave pattern is modeled as:

$$\text{Pattern}(t) = A \cdot \text{sign}\left(\sin\left(\frac{2\pi t}{P}\right)\right)$$

Where A and P follow the same distributions as the sine pattern.

Triangle Pattern: The triangle wave pattern is modeled as:

$$\text{Pattern}(t) = A\left(2\left|\frac{t}{P} - \left\lfloor\frac{t}{P} + 0.5\right\rfloor\right| - 1\right)$$

Where A and P follow the same distributions as the sine pattern.

Sawtooth Pattern: The sawtooth wave pattern is modeled as:

$$\text{Pattern}(t) = A\left(2\left(\frac{t}{P} - \left\lfloor\frac{t}{P}\right\rfloor\right)\right)$$

Where A and P follow the same distributions as the sine pattern.

B.3 Noise Component

The noise component models random fluctuations and can follow one of two distributions:

Gaussian Noise: The Gaussian noise is modeled as:

$$\text{Noise}(t) \sim \mathcal{N}(\mu, \sigma^2)$$

Where μ is the mean and σ is the standard deviation, both sampled from uniform distributions:

$$\mu \sim U(\text{mean}_{\text{min}}, \text{mean}_{\text{max}}), \quad \sigma \sim U(\text{stddev}_{\text{min}}, \text{stddev}_{\text{max}})$$

Uniform Noise: The uniform noise is modeled as:

$$\text{Noise}(t) \sim U(\text{low}, \text{high})$$

Where low and high are the lower and upper bounds of the uniform distribution, both sampled from uniform distributions:

$$\text{low} \sim U(\text{low}_{\text{min}}, \text{low}_{\text{max}}), \quad \text{high} \sim U(\text{high}_{\text{min}}, \text{high}_{\text{max}})$$

B.4 Final Time Series

The final time series $\{x_t\}_{t=1}^{T}$ is computed as the sum of the selected trend, pattern, and noise components. Each component may be absent, in which case the corresponding term is set to zero.

$$x_t = (\text{Trend}(t) \text{ if present}) + (\text{Pattern}(t) \text{ if present}) + (\text{Noise}(t) \text{ if present})$$

C Appendix: Pattern Type Distribution with Varying Configurations

This appendix contains the plots for pattern type distributions and other related attributes for the synthetic data generated with different configurations. The only variation across the configurations is in the *pattern type*, while all other parameters remain the same. The following configuration was used for generating the data:

```
n_series: 512
length: 512
trend_types: ['none']
pattern_types: ['triangle', 'none', 'sine']
noise_types: ['none']
trend_params:
  slope: [0.01, 0.1]
  intercept: [-5, 5]
  growth_rate: [0.01, 0.1]
pattern_params:
  amplitude: [-1, 1]
  period: [128, 164]
```

C.1 Experiment: Constant & Sine Patterns

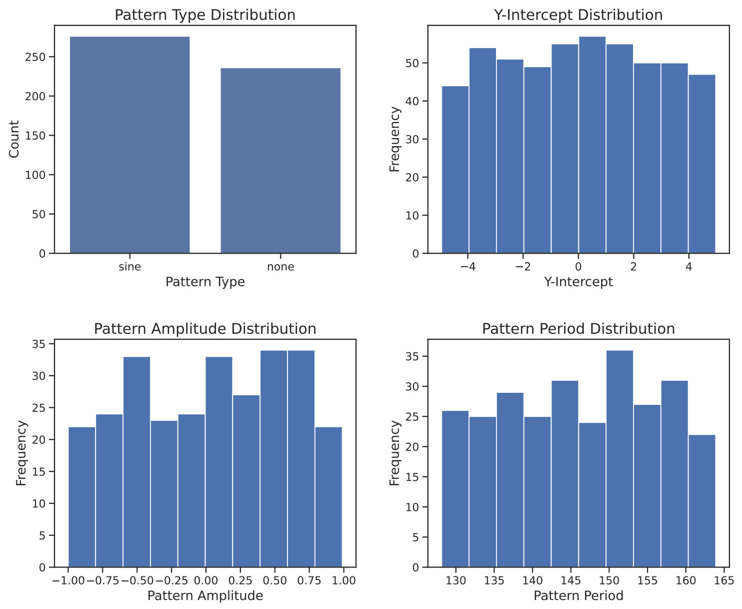


Figure 5: Pattern type distribution and related attributes for the configuration where patterns are none (constant) or sine.

C.2 Experiment: Constant & Triangular Patterns

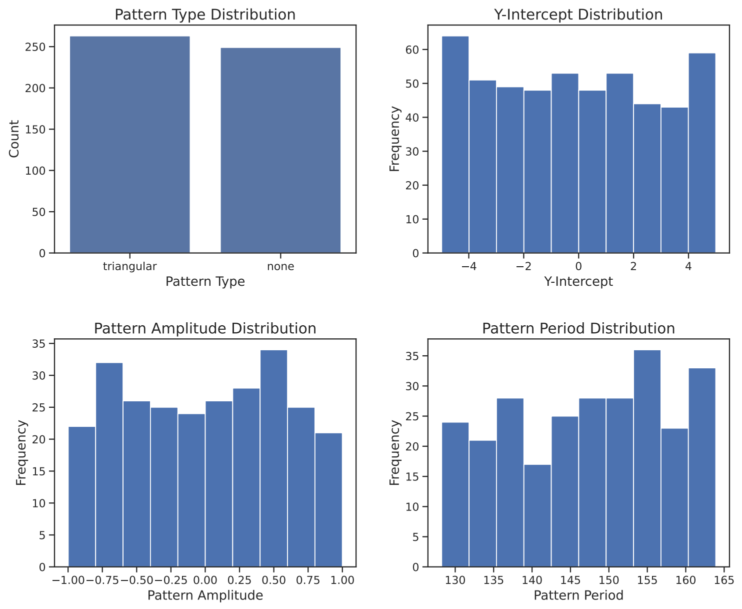


Figure 6: Pattern type distribution and related attributes for the configuration where patterns are none (constant) or triangular.

C.3 Experiment: Sine & Triangular Patterns

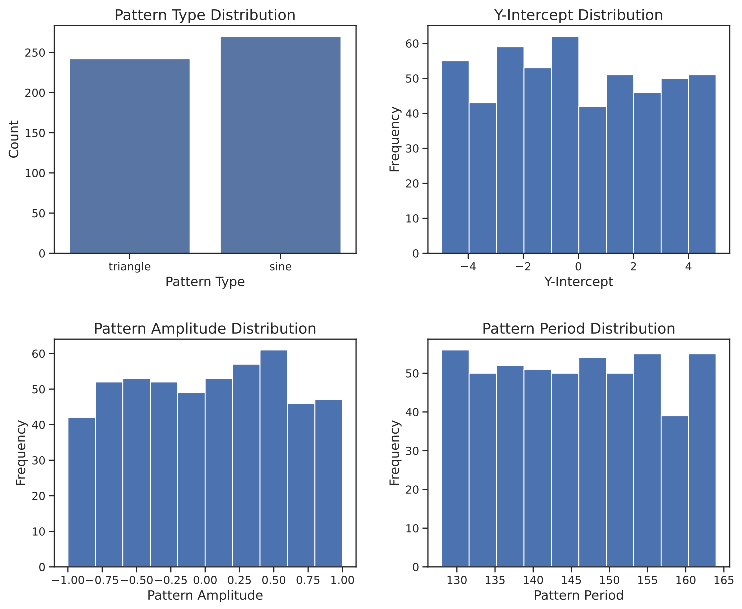


Figure 7: Pattern type distribution and related attributes for the configuration where patterns are sine or triangular.

D Appendix: Constant-to-sine Steering Matrix Generalization

We want to assess the generalization of our steering matrix which we obtained in a **constant-to-sine** setup we considered earlier. To do this, we check if inputs with **increasing and decreasing** trends with **linear and exponential** growth will get altered into a sine-like series at the output.

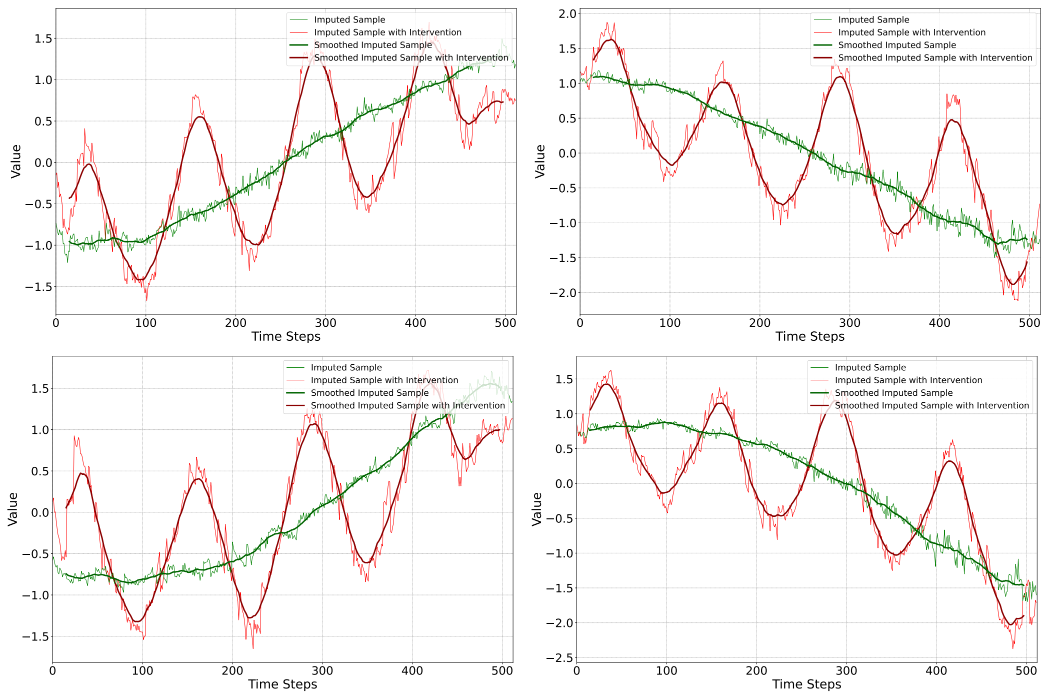


Figure 8: **Constant-to-sine steering matrix generalization** for other input trends: (Top left) increasing linear, (Top right) decreasing linear, (Bottom left) increasing exponential, and (Bottom right) decreasing exponential.

E Appendix: Linear Separability of Features

E.1 Constant vs Sine Pattern

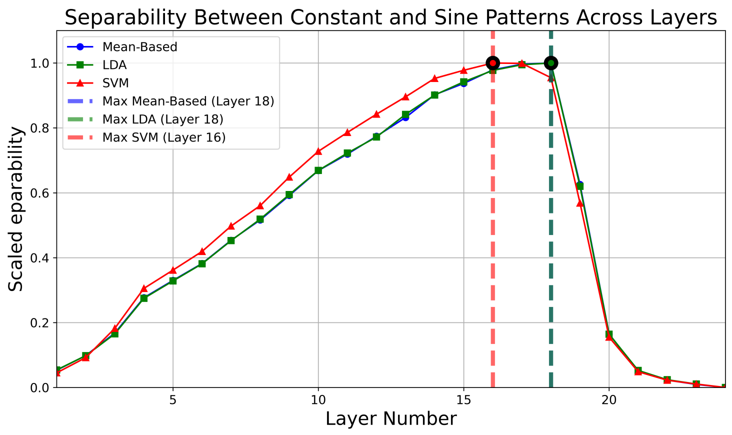
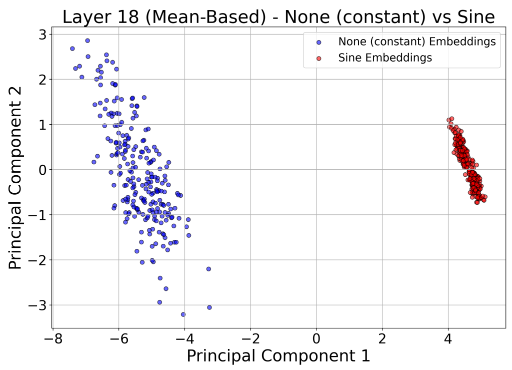


Figure 9: Scaled separability for different types of linear discriminators across the layers of the model for constant vs sine pattern setup.



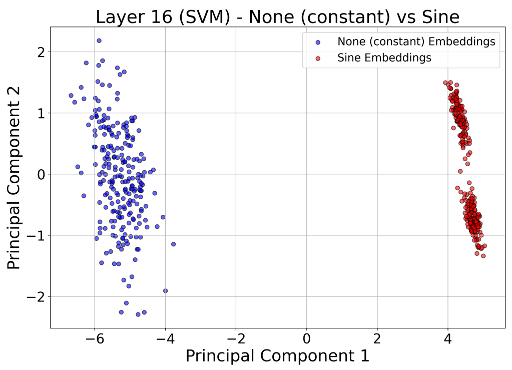


Figure 10: Comparison of constant and sine patterns in the embedding space across the layers with maximum separability. The layer with max separability for LDA and mean-based methods is the same (layer 18).

E.2 Constant vs Triangular Pattern

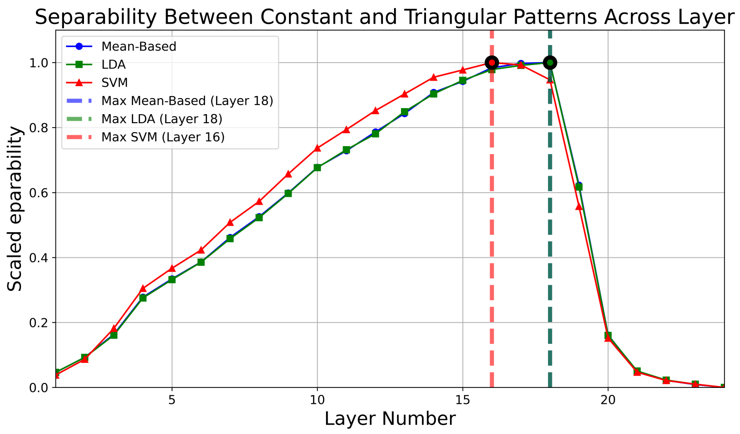
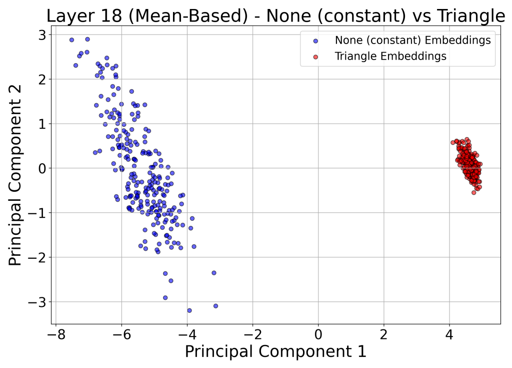


Figure 11: Scaled separability for different types of linear discriminators across the layer of the model for constant vs triangular pattern setup.



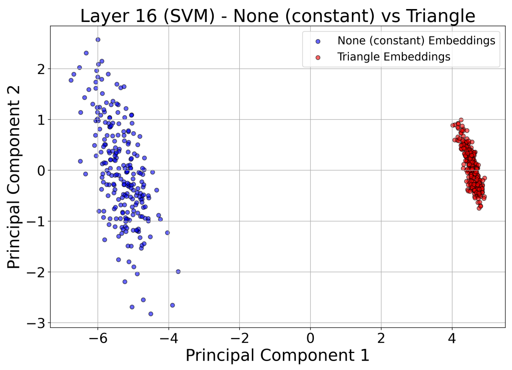


Figure 12: Comparison of constant and triangular patterns in the embedding space across the layers with maximum separability. The layer with max separability for LDA and mean-based methods is the same (layer 18).

E.3 Sine vs Triangular Pattern

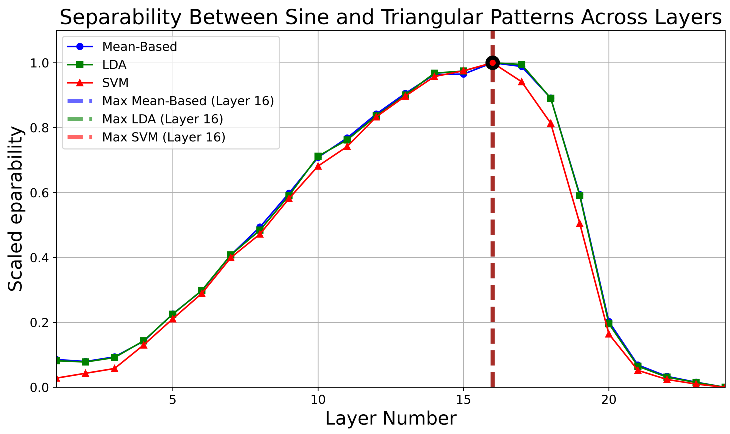


Figure 13: Scaled separability for different types of linear discriminators across the layer of the model for sine vs triangular pattern setup.

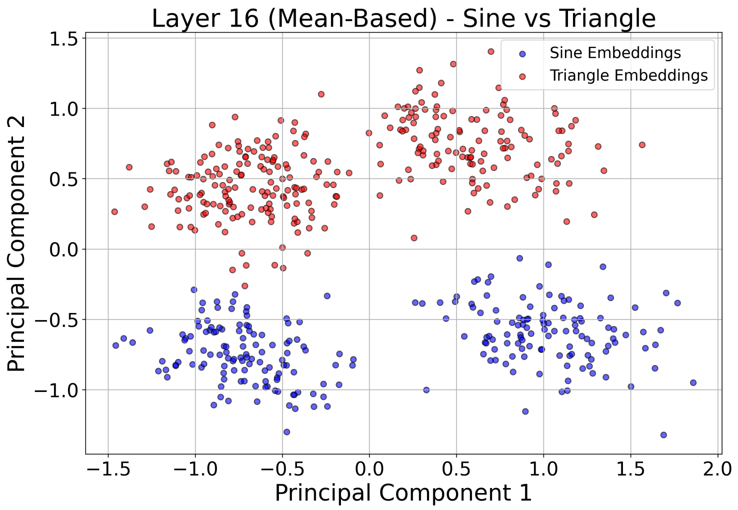


Figure 14: Comparison of sine and triangular patterns in the embedding space across the layers with maximum separability - max separability for all methods is the same, that's why we show only the mean-based plot.