Building a GPT for System Identification

Mohammed Mohammed Ahmed

Master of Science
School of Informatics
University of Edinburgh
2024

Abstract

System identification for Dynamical systems and biological data has been a focus of research for decades, the recent advancement in Artificial Intelligence driving significant progress in the field. Traditional methods often struggled with long-term forecasting, generalizability, and interpretability thus limiting their practical application. In this study, we explore the use of a repressilator regulatory network, this network has been used in many applications of synthetic biology including oscillatory gene circuits, metabolic engineering, drug delivery and mathematical modelling. Recent developments in Koopman theory have shown a better ability to find correct Koopman observables and Koopman operator using machine learning. Generative Pretrained Transformers (GPT) have shown promise in system identification. Combining these approaches has improved long-term forecasting of the generalizability aspects. Additionally, the introduction of the Kolmogorov-Arnold Network (KAN) offers a new avenue for enhancing interpretability and approximation accuracy.

We investigate various architectural designs for integrating KAN networks within the Koopman embedding framework, this includes combining MLP and KAN as well as configuration where MLP and KAN are used separately or in different parts of the network. Detailed architectural considerations were necessary for effective KAN implementation in each model. These architectures aim to leverage KAN for different tasks within the Koopman embedding framework where in the future it can be used for interpretation of the underlying mathematical functions. Furthermore, we conducted an in-depth analysis to understand the impact of Koopman embedding on GPT training, providing insights into how embedding models influence GPT performance.

Keywords: Koopman Embedding, Kolmogorov-Arnold Networks, Repressilator, Generative Pretrained Transformer

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Mohammed Mohammed Ahmed*)

Acknowledgements

First and foremost I would like to thank my supervisor, Professor Filippo Menolascina, for his invaluable guidance, insightful discussions, and constructive feedback throughout this project. A special thanks to Evangelos Nikolados for his insightful conversation in navigating the experimental aspects of this research. Furthermore, I would like to acknowledge my colleagues, the university, and my scholarship sponsor, Google DeepMind, for their support throughout this academic year.

Finally, I want to express my sincere gratitude to my family and friends for their continuous support and motivation during this year. Their unwavering support and belief have been invaluable to me throughout this year, without them I could not have done this.

Table of Contents

Chapter 1

Introduction

Dynamical systems are widely used to mimic a wide range of phenomena in a variety of domains, including physical systems like pendulum motion [26], biological systems such as disease transmission [37], and economic systems. This field led to the development of numerous techniques for system identification tailored for different tasks, including linear and nonlinear system identification, data-driven methods, Gaussian system identification, and more. Data-driven methods like Koopman [4], Dynamic Mode Decomposition [34], and SINDY model [5], data-driven models can even be employed to evaluate and validate existing models by comparing mathematical representation against data-driven results. However, these methods have different challenges such as the difficulty of finding a finite set of observables for identification, finding linear transformation for non-linear systems, sensitivity to noise and data quality, generalizability even to model with small changes in parameters, and lastly interpretability especially when machine learning is utilized where we can't infer the mathematical model behind it.

Recent advancements in Natural Language Processing (NLP) have utilized transformer architectures that generalize for a wide range of tasks by learning from vast amounts of data, this is extended to other fields such as Computer Vision and recently in time series forecasting. In system identification, [12] have used a transformer model applied to the Wiener-Hammerstein system with simple nonlinearity between two linear time-invariant (LTI) systems, achieving results comparable to traditional model fitting approaches. To address the limitations of DMD and Koopman in identifying finite invariant subspaces, researchers [13] have developed a GPT-based model that builds on Koopman dynamics to accurately model physical systems like the Lorenz system.

In biological systems, model accuracy and interpretability are crucial. Applications

of dynamical systems in biology include modelling Gene Regulatory Networks, Population Dynamics, and physiological systems, where the focus is on creating reliable and interpretable models that can provide insights into complex biological processes.

1.1 Problem statement

This project aims to develop and refine methods for identifying and predicting the future state of biological systems. The main challenge lies in the nonlinear behaviour of biological systems, which requires advanced modelling techniques. In our experiments, we will use the Kolmogorov-Arnold Network (KAN) [24] network that addresses the interpretability and design a model that doesn't affect the accuracy. We will use Koopman embedding architecture as a preliminary step for the Generative Pretrained Transformer (GPT) model. Additionally, we will explore various Koopman autoencoder architectures and analyze how they might affect GPT training and investigate whether foundational models for time series can be adapted for system identification. Finally, we aim to identify ways to construct an interpretable Koopman system.

1.2 Contribution

In this work, we developed multiple Koopman autoencoder networks. To address the ability for interoperability, we created architectures that could use either Multi-Layer Perceptron (MLP) or KAN network or a combination of both. KAN network uses the B-spline function on edges and the Kormoglov-Arnold theory for function approximation. We explored various combinations of these two techniques to assess embedding ability and evaluate how each model performs before integrating them with a GPT model. Additionally, we designed a framework for Koopman embedding with operator parameterization which is capable of utilizing either KAN alone or a mixture of KAN and MLP for the Koopman operator.

1.3 Thesis structure

his thesis is organized into five parts:

1. Introduction: The first chapter introduces the project's motivation, goals, and contribution, providing an overview of the project.

2. Background: This chapter covers the foundational knowledge necessary in this research, including prior work in time series analysis using transformers. It aims to familiarize readers with the project and enhance their understanding of the work presented.

3. Methodology: This chapter outlines the project's implementation steps in detail, describing the overall structure, pipeline construction, and various evaluation methods used.

4. Experiments and Results: Here, the experimental setup is presented, along with the different architectural designs and the results obtained from the embeddings and GPT training.

5. Summary: The final chapter summarizes the project's contributions and key findings.

Chapter 2

Background

2.1 System Identification and Machine Learning

System identification introduced by Zadeh [48] is a framework for constructing mathematical models for dynamical systems based on measurements of input and output signals. This area has undergone significant theoretical development, including evolving subfields of control theory with numerous applications in engineering and science [18]. Meanwhile, machine learning has emerged as a leading research field, that focuses on developing algorithms that learn from data and make predictions without explicit programming. Both system identification and machine learning share the goal of inferring models from observed data, the two approaches share several important links and considerations between these two paradigms when applied.

the steps for constructing a system identification flow figure 2.1 are as follow :

1. Experiment design: Experiments are conducted to gather data that should accurately characterize the system behaviour.
2. Model Structure: A suitable model structure is chosen using preliminary analysis and prior knowledge of the system.
3. Fitting Criteria: A cost function is selected to reflect how the model fits the data.
4. Parameter estimation: An optimisation problem is solved to obtain the numerical values of the model parameters.
5. Model Validation: Testing the model on unseen data to reveal any inadequacies.

Identifying the appropriate model structure is a crucial aspect of system identification. One way to conceptualize the model is through the predicted input-output

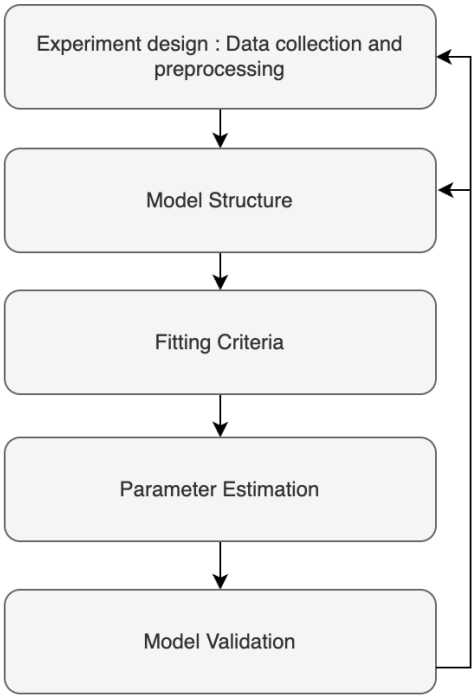


Figure 2.1: Procedure of system identification

relationships, represented as $y(t|\theta)$, where θ is the model parameter. According to [25], machine learning models can be regarded as types of system identification models. This is evident through the analogy where two system identification techniques are identified:

NARX Model: The Nonlinear Autoregressive with Exogenous Inputs (NARX) model uses a static nonlinear function of a regression vector $\varphi(t)$ formed by input-output observations:

$$\hat{y}(t|\theta) = f(\varphi(t), \theta) \tag{2.1}$$

This is related to deep neural networks, where the model serves as a function approximation based on input observations. For a one-layer neural network, the function with an activation function is defined as:

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{x} + b)$$

Here, σ is a nonlinear activation function, while w and b are the weight matrix and bias, respectively. A multi-layer neural network, or multi-layer perceptron (MLP), consists of multiple layers where each layer applies a linear transformation followed by a nonlinear activation function. Denoting the one-layer function as F, the function for an L-layer network is expressed as:

$$y = \mathrm{F}_L(\mathrm{F}_{L-1}(...\mathrm{F}_2(\mathrm{F}_1(x)))) \tag{2.2}$$

Various architectures of multi-layer neural networks are detailed in [14]. It is important to note that all the deep networks described can be considered as NARX models.

NLSS Models: Nonlinear state space models NLSS Models is formed by the following equation:

$$x(t+1) = f(x(t), y(t), u(t), \theta) \tag{2.3}$$

$$\hat{y}(t \mid \theta) = h(x(t), \theta) \tag{2.4}$$

where f and h are parametrized by θ. In contrast to Feedforward networks, the NARX model, which can only propagate information to the next time step from a finite sequence of past inputs and outputs will inherently limit their ability to model temporal dependencies beyond fixed window size. The Recurrent Neural Networks (RNNs) address the limitation of Feedforward NN by introducing the concept of memory, where the architecture will allow to maintain the hidden state from the previous time steps. The hidden state is updated at each time step and is used along with the current system input to produce the output. The update equation is given by:

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \tag{2.5}$$

$$y_t = W_{yh}h_t + b_y \tag{2.6}$$

where h_t represents the hidden state at time t, even though RNNs theoretically can capture information for an arbitrary number of time steps they struggle with long-term dependencies due to vanishing and exploding gradient problems. Long Short-Term Memory (LSTM) networks are an enhanced form of RNNs designed to overcome the limitations of standard RNNs in handling long-term dependencies. Long Short-Term Memory (LSTM) networks are an improved version of RNNs that address the limitations of ordinary RNNs in dealing with long-term dependency. LSTMs incorporate memory cells and gating mechanisms, allowing them to selectively remember or forget information over long sequences.RNNs and LSTMs closely resemble NLSS models because they both capture temporal relationships and describe system dynamics. In particular, LSTMs are a type of NLSS model in which the state is represented by the cell state c(t) and the hidden state h(t). This relationship emphasises the utility of RNNs and LSTMs in system identification tasks that require capturing nonlinear dynamic characteristics.

2.2 Preliminaries of Transformer

The transformer architecture, introduced by [40] shown in figure 2.2, was developed to address the limitations of RNNs and LSTMs in sequence modeling within the encoder-decoder framework. The sequential nature of these models hinders parallelization during training, particularly for longer sequences. In contrast, the transformer exclusively employs an attention mechanism to capture global dependencies between input and output, eliminating the need for recurrence. Each encoder block comprises a multi-head self-attention layer and a position-wise feed-forward network. Decoder blocks incorporate both multi-head self-attention and cross-attention modules, followed by a position-wise feed-forward network. The following parts are essential in understanding how the transformer works:

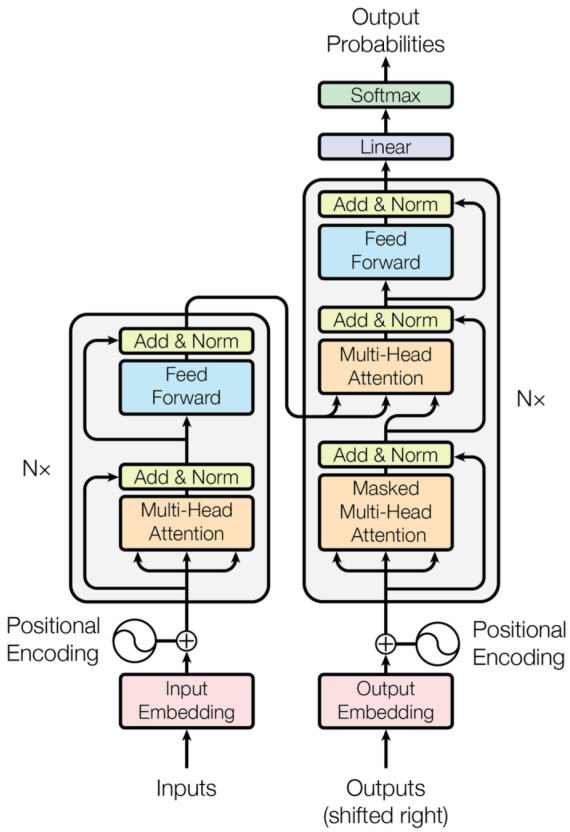


Figure 2.2: The Transformer - model architecture.

2.2.1 Positional Encoding:

The transformer architecture utilizes the positional encoding added to the input encoding to model the sequential information passed to the encoder and the decoder modules. In the original paper, they used the absolute positional encoding with the sinusoidal function to calculate the positional embedding in the equation below they hypothesised that this might also allow the model to learn the relative position.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{2.7}$$

Another two forms of positional encoding are relative and learnable positional encoding. In relative positional encoding captures the relative distances between tokens rather than their absolute positions. This approach focuses on the relationships between tokens, which can be more beneficial for certain tasks [35] showed the benefit of this approach is that it's more flexible and generalizes well for different sequence lengths but may add additional complexity to the model. The learnable positional encoding of each position in the sequence is assigned a learnable embedding vector. These vectors are initialized randomly and optimized during training, similar to token embedding.

2.2.2 Multi-head Attention

One of the main components of the transformer architecture is the multi-head attention module, which enables the transformer to capture the intricate relationships between various input sequence elements at the same time. Unlike single-head attention, multi-head attention performs multiple functions or heads in parallel, which leads to the capture of different features, improved representation, and increased model capacity. with Query-Key-Value matrices we calculate the scaled attention as:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.8}$$

where queries $\mathbf{Q} \in R^{N\times d_k}$, keys $\mathbf{K} \in R^{M\times d_k}$, values $\mathbf{V} \in R^{M\times d_v}$, d_k, d_v are the dimensions of keys (or queries) and values, N, M are the lengths of queries and keys(and values). In multi-head attention the attention is calculated as follows:

$$\text{MultiHead}(Q,K,V) = \text{Concat}(head_1,...,head_h)W^O \tag{2.9}$$

where $head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$.

2.2.3 Residual Connection, Layer Normalization and Feed Forward Network

In the encoder and decoder models, a feed-forward network is applied after the input has passed through the attention module. This adds another level of complexity by applying non-linear functions to these representations and also refines the features extracted by the self-attention layer by highlighting significant features and suppressing less relevant ones. There is a residual link that comes after each multi-head attention or feed-forward neural network instance, and layer normalisation follows next. The layer normalisation facilitates faster and more stable training, while the residual connection helps prevent the vanishing gradient problem.

2.3 Transformers in Time Series

The Transformer ability to model long-range dependencies within sequential data has spurred its application to time series forecasting. A diverse range of Transformer architectures has been proposed to address various forecasting challenges, including prediction [49], classification [22], and anomaly detection [39]. This adaptability is attributed to Transformer ability to effectively capture both long-term and short-term temporal patterns, as well as seasonal components. The taxonomy presented in Figure 2.3 is based on the survey paper [41]. At the modular level, numerous variations of the vanilla Transformer architecture have emerged through developing novel attention mechanisms, alternative data normalization strategies, and incorporating bias terms in token embeddings (e.g., segmentation-based representation, and cross-dimensional dependencies).

The emergent advances in Large Language models with their ability to solve various complex tasks by in-context learning, these models are categorized by [17] as embedding-visible LLMs which are usually open-sourced with publicly accessible inner states adding the ability to fine-tune and good result in few-shot and zero-shot without additional training and example of these models are BLOOM [29], Llama [38], and Vicuna [9]. The second type is embedding- invisible LLMs which are closed sources with invisible inner states such as GPT-4 [1] and Gemini [28] these types of models are only accessible by API calls.

Current research tries to utilize LLM for time-series forecasting we can differentiate these emergent models into general-purpose time-series forecasting and domain-specific

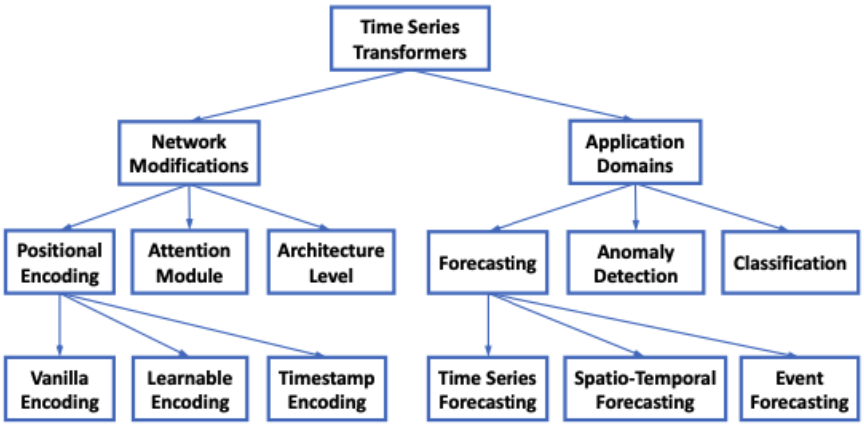


Figure 2.3: Taxonomy of Transformers for time series modelling from the perspectives of network modifications and application domains.

forecasting. General Models such as Time-LLM [16] where input time series are reprogrammed into more natural text prototypes and augmenting the input with declarative prompts (e.g., domain expert knowledge and task instructions) are also added to help LLM reasoning which led to achieving state-of-the-art performance in various forecasting scenarios and excelling in both few-shot and zero-shot settings, TEMPO [7] have employed another technique by decomposing the complex interactions among trend, seasonal, and residual components, and introducing the design of prompts to facilitate distribution adaptation across different types of time series. In the field of healthcare and bioinformatics, [23] have demonstrated that the limitations of LLMs regarding numerical data can be addressed with few-shot prompt tuning. In healthcare, it is crucial for insights to be based on quantitative measurements, with datasets derived from wearable and clinical-grade sensing devices. This approach leads to significant improvements over zero-shot inferences and supervised baselines in health-related tasks.

Pre-trained foundation models (PFMs) are large-scale models capable of handling a wide range of downstream tasks. LLMs are essentially a subset of PFM that is employed for natural language tasks. Since 2022 multiple research have been done on the foundational model for time series, TS2VEC [47] where the authors used a unified framework to learn contextual representation for sub-series at different semantic levels. They used hierarchical contrasting in both instance-wise and temporal dimensions to capture multi-scale contextual information, and contextual consistency for positive pair selection, which is better suited for time series data with differing features. [43] To

address the challenges of forecasting data with multiple frequencies, multiple variables, and varying distributions, the authors employed several techniques: using different patch sizes to handle varying frequencies, flattening multivariate time series into a single sequence, and modelling data distribution with a mixture model. Lag-llama [32] is a foundation model for univariate probabilistic forecasting that tokenizes the time-series in a general way without relying on a frequency from a specific dataset which generalizes better on unseen frequencies at test time. MOMENT [15] have addressed the issue of a lack of large public time series repositories by assembling a large and diverse collection of public time series known as the Time series Pile, and have built on recent work to design a benchmark for evaluating time series foundation models on diverse tasks and datasets in limited supervision settings.

2.4 Koopman Operator Theory

The Koopman operator [4] demonstrated that it's possible to represent a nonlinear dynamical system in a linear infinite dimensional measurement on Hilbert space. This spectral decomposition completely characterises the behaviour of the nonlinear system; however, operating in an infinite dimension poses a new challenge, so the research focused on obtaining a finite approximation of the koopman operator, which will help us in utilising optimal optimization and control available for linear systems.

The Koopman operator K_t is an infinite-dimensional linear operator that acts on measurement functions g known as *observable*:

$$\mathcal{K}_t g = g \circ \mathbf{F}_t \tag{2.10}$$

in discrete time this function can be represented as:

$$\mathcal{K}_{\Delta t}(g(x_k)) = g(\mathbf{F}_{\Delta t}(x_k)) = g(x_{k+1}) \tag{2.11}$$

This can be used to the model evolution through time from timestep t onward by the function:

$$g(x_{t+1}) = Kg(x_t), \quad g(x_{t+2}) = K^2 g(x_t), \quad g(x_{t+3}) = K^3 g(x_t), \ldots, \quad g(x_{t+n}) = K^n g(x_t)$$

While simplifying analysis, it necessitates careful selection of observable functions to accurately represent the system. Spectral analysis of the Koopman operator reveals

the underlying dynamical modes. Essentially, there is a trade-off between simpler dynamics in the lifted space and the difficulty of finding relevant observables.

Recent research has leveraged machine learning to effectively learn finite Koopman representations for dynamical systems [26]. Various techniques have been applied, such as autoencoders in [26], kernel methods in [42], and optimization algorithms proposed by [36]. Although deep learning methods have improved the discovery of Koopman observables and operators, these approaches have not yet been demonstrated to be effective for the long-term prediction of high-dimensional systems. This limitation is due to challenges such as approximating finite-dimensional Koopman observables, limited data availability, and complete reliance on the discovered Koopman operator K to model the dynamics. Predicting a system through a single linear transformation has significant limitations and can be considered a naive approach from a machine-learning perspective.

2.5 Kolmogorov–Arnold Networks

Kolmogorov-Arnold Networks (KAN) [24], inspired by Kolmogorov–Arnold representation theorem [19]. KAN is considered a new neural network (NN) type that uses a different approximation theorem compared to Multi-Layer Perceptron (MLP) based on the universal approximation theorem. MLPs have static predefined activation functions on their nodes ”neurons”, but KANs have learnable activation on their edges ”weights” between nodes and have no linear weights which are replaced by a univariate function parameterized as a spline. Mathematically KAN function can be represented by:

$$f(x_1, x_2, ..., x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$$

Here, $\phi_{q,p}$ are our univariate function learned during training, and Φ_q akes the univariate functions and combines them. To determine these univariate functions, B-splines are employed. B-splines are an advanced curve-fitting technique and a specific type of mathematical function characterized as flexible, piecewise-polynomial, and capable of defining a smooth curve through a set of points. Mathematically, we can define a b-spline as:

$$\text{spline}(x) = \sum_i c_i B_i(x)$$

where c_i represents the spline coefficient. These are weights assigned to each basis function $B_i(x)$ and will determine the shape of the spline, $B_i(x)$ are predefined function that forms the basis for constructing the spline such as piecewise cubic polynomial. In addition to techniques used in MLPs, such as increasing the depth and width of layers, the KAN network employs a grid extension approach. This method allows a spline to approximate a target function with arbitrary accuracy by making the grid increasingly fine-grained. The figure from the original paper [24] below illustrates the many optimisation steps and the operation of the KAN network.

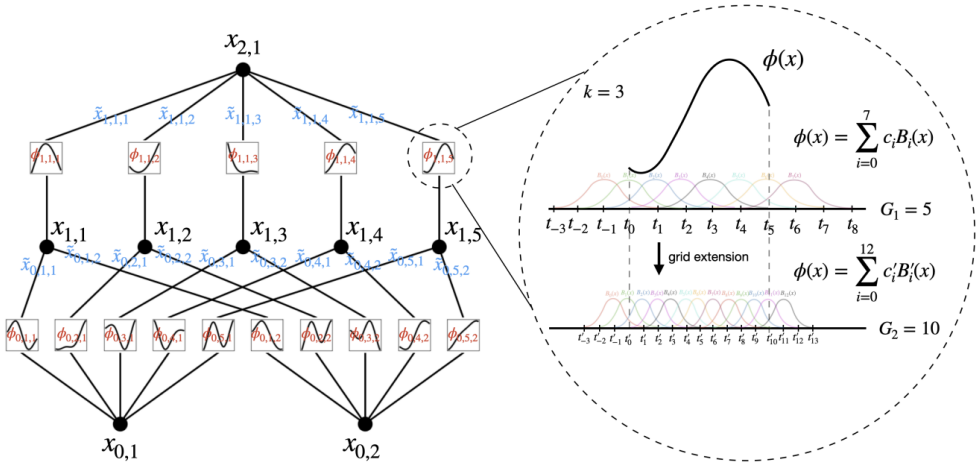


Figure 2.4: On the left is the network's activation flow. On the right, a B-spline is used to parameterise an activation function that can be more coarse-grained grids for better approximation.

2.6 Repressilator Model

The repressilator [11] is a genetic regulatory network characterized by a feedback loop involving three genes. Each gene produces a protein that inhibits the expression of the subsequent gene in the cycle. This negative feedback loop results in a dynamic system that oscillates over time. The mathematical model of the repressilator can be

represented as:

$$\begin{aligned}
\dot{m_0} &= -m_0 + \frac{\alpha}{1+p_2^n} + \alpha_0 \\
\dot{m_1} &= -m_1 + \frac{\alpha}{1+p_0^n} + \alpha_0 \\
\dot{m_2} &= -m_2 + \frac{\alpha}{1+p_1^n} + \alpha_0 \\
\dot{p_0} &= -\beta(p_0 - m_0) \\
\dot{p_1} &= -\beta(p_1 - m_1) \\
\dot{p_2} &= -\beta(p_2 - m_2)
\end{aligned} \tag{2.14}$$

where m_i represents the concentration of mRNA, p_i denotes the concentration of the protein, α is the basal transcription rate, α_0 indicates the number of protein copies produced from a given promoter type during the continuous growth while $\alpha + \alpha_0$ during its absence, β represents the ratio of the protein decay to the mRNA decay rate, and n is the Hill coefficient. In the phase portrait, the genetic circuit displays a limit cycle, with a single basin of attraction centred around the origin. We concentrate on a three-dimensional model simplified from the initial experimental implementation of the repressilator [11] in our one-parameters dataset ($\alpha = 1000, \alpha_0 = 1, \beta = 5, n = 2$). This model captures the limit cycle and basin of attraction, allowing us to study the effect of initial conditions -as shown in Figure - on the performance evaluation of the nonlinear system.

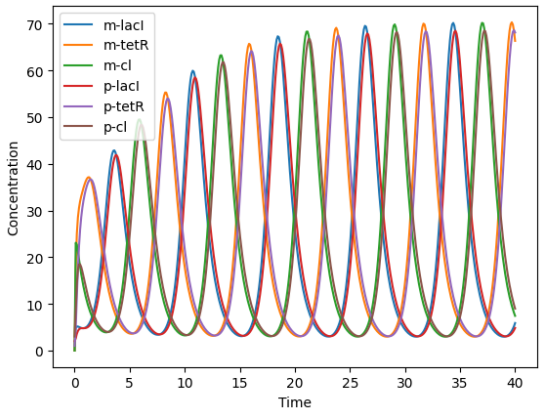


Figure 2.5: y = [0, 0, 0, 2, 1, 3]

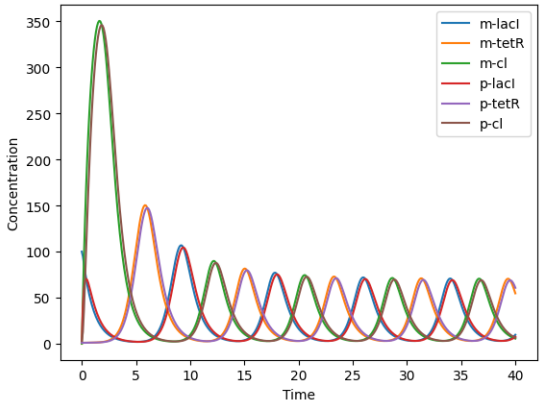


Figure 2.6: y = [100, 0, 0, 2, 1, 3]

Figure 2.7: Repressilator Model for two initial conditions

Chapter 3

Methodology

3.1 Embedding Model

For the embedding model, we will utilize the approximation of Koopman dynamics using the autoencoder model shown in figure 3.1, where the encoder model will approximate the Koopman observable function using a neural network and a predefined observable dimension in which $\mathcal{F}(x_k) \approx g(x_k)$.

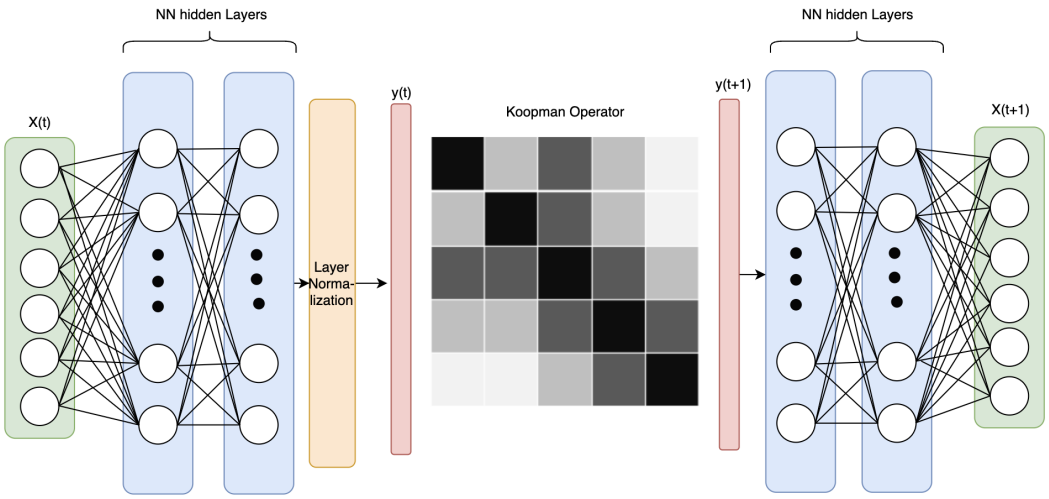


Figure 3.1: Enter Caption

Building upon the work of [13] and [20] Koopman operator assumes the form of a learnable banded matrix that is optimized with the auto-encoder. [13] claimed that this reduction of the learnable parameter in the Koopman operator helped better discover the dynamical modes prevent the model from overfitting to high-frequency fluctuations and allow training embedding with higher dimensionality. the loss function for the

embedding model is a weighted sum of three parts:

- The reconstruction loss ensures consistent mapping from embedding space $\mathcal{L}_{recon} = ||x(t) - \mathcal{G}(\mathcal{F}(x(t)))||$ where $\mathcal{G}$ is the inverse function in the decoder
- The dynamics loss which pushed the Koopman observable to follow linear dynamics $\mathcal{L}_{dynamics} = ||x(t+1) - \mathcal{G}\mathcal{K}(\mathcal{F}(x(t))||$
- The decay of the Koopman operator helps in preventing overfitting and helps the model to discover meaningful dynamical modes $\mathcal{L}_{decay} = ||\mathcal{K}||_2^2$

where $||.||$ is the mean square error (MSE), the combined loss equation can be represented as:

$$\mathcal{L} = \alpha_1 \mathcal{L}_{recon} + \alpha_2 \mathcal{L}_{dynamics} + \alpha_3 \mathcal{L}_{decay} \tag{3.1}$$

In Koopman embedding, the loss function is designed to assign similar embeddings to time steps that are either close in time or share the same underlying dynamics. Since our goal is not to precisely identify the ideal Koopman dynamics, but rather to apply Koopman theory to reinforce physical context through learned dynamics, we chose to increase the weight of α_2 compared to other loss terms. This ensures that our model achieves a strong reconstruction loss, which is crucial for the GPT model to accurately capture the correct dynamics. This approach is akin to NLP embeddings like context2vec [27] and ELMo [30], where word context and associations are used to map related words to similar embeddings.

3.1.1 MLP vs KAN

The original KAN paper [24] has demonstrated that KAN outperformed MLP in tasks involving PDEs and data fitting and doing so by having fewer parameters. However, as the authors note, the original implementation of KAN is significantly slower to train than MLP, and only comparing them on symbolic function representation might be unfair since KAN utilises the B-spline function, leading to better formula representation. Several other papers have adapted KAN for various machine learning applications, such as computer vision [8], time series analysis [45], and Physics Informed Neural Network [44]. [46] provides a fairer comparison between KAN and MLP across different machine learning tasks, including audio processing, computer vision, NLP, and symbolic function representation while controlling for the same number of parameters and FLOPs. In this comparison, KAN is outperformed by MLP except for formula

representation, for KAN to outperform MLP it typically requires more computational resources.

Recent advances in KAN networks have highlighted differences in performance and model interpretability compared to traditional MLP networks, leading to distinct architectural choices. In the context of Koopman embedding, the interpretability of KAN networks provides the advantage of potentially uncovering the mathematical transformation of observables and dynamics. MLP networks, while simple and easy to train may struggle with limited expressivity due to their fixed activation functions. This limitation require the use of deeper networks to model complex physical dynamics effectively which in turn harms interpretability. On the other hand, KAN networks are designed to capture intricate nonlinear dynamics, but as noted by the authors they require careful hyperparameter tuning and architectural design consideration.

We developed five autoencoder architectures combining MLP and KAN networks to test KAN's effectiveness as an embedding model. Given the potential of KAN network to infer activation function in neural networks, we aim to ensure it can be used in a more complex application, extending beyond the original paper. This architecture helps determine whether KAN can effectively capture the dynamic and potentially infer the transformation and recovery function between input and embedding spaces. The description of the architecture are as follows:

Detailed Descriptions of Architectures:

1. MLP Architecture: Both the encoder and decoder employed MLP networks for Koopman embedding.

2. KAN Architecture: Both the encoder and decoder utilized KAN networks for Koopman embedding.

3. eKdM Architecture: The encoder used a KAN network to generate interpretable Koopman observables, while the decoder was an MLP for the reverse mapping.

4. eMdK Architecture: The encoder was an MLP, and the decoder employed a KAN network.

5. MLP+KAN Architecture: This hybrid architecture combined MLP and KAN networks for both encoder and decoder. The MLP preprocessed input data to match the B-spline range, optimizing KAN parameter efficiency.

3.2 Transformer Decoder

The following stage of the pipeline involves feeding the embedded dynamical system time series into the decoder model. This embedded sequence is represented as $\mathcal{E} = [e_i, e_{i+1}, ..., e_n]$ augmented with positional encoding as in section 2.3. Consequently, as illustrated in Figure 3.2, the proposed architecture is a simplified GPT2 [31] variant tailored to this task compared to its NLP counterpart where our architecture is more simple in terms of the number of decoder layers. To train the GPT model on a dataset of diverse time series originating from varying initial conditions denoted as $D = \mathcal{E}_{i=1}^{n}$ that adhere to the concept of independent and identically distributed, the log-likelihood for one-step-ahead prediction is expressed as follows:

$$L_D = \sum_{i=1}^{D} \sum_{j=1}^{T} -\log p\left(e^i_j \mid e^i_{j-k}, \ldots, e^i_{j-1}, \theta\right) \tag{3.2}$$

In contrast to NLP tasks where output tokens are determined via softmax, our model employs a standard Gaussian loss function between the transformer's prediction and the input embedding. As each point in the embedding space corresponds to a specific value in the continuous input measurement domain, the model parameters are denoted by θ, and the context window for the GPT model is represented by K.

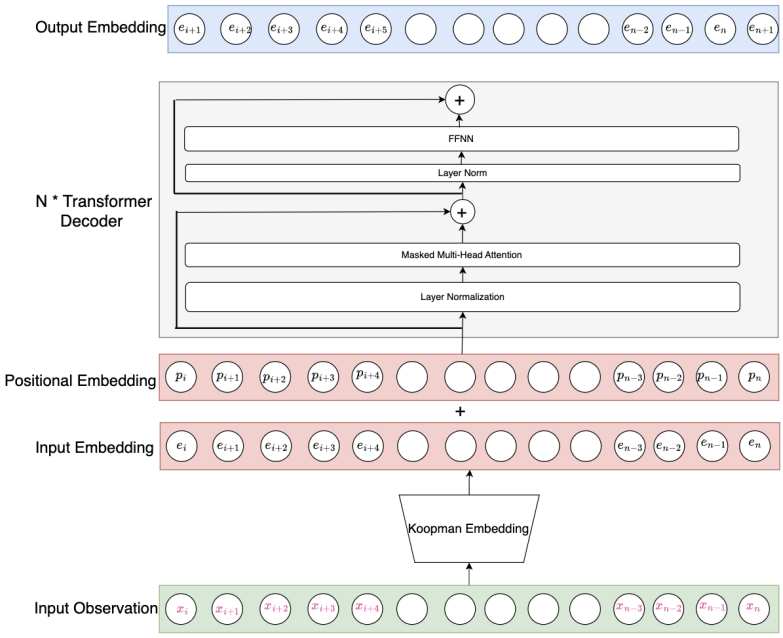


Figure 3.2: The Transformer Decoder uses Koopman Embedding to model dynamical systems

[13] have identified a close relationship between the self-attention mechanism and numerical time-integration methods used in dynamical systems. their theorem states that Consider a dynamical system of the form $\frac{d\phi}{dt} = f(t,\phi), \quad \phi(t) \in R^d$ where $f : R \times R^d \to R^d$ is Lipschitz continuous with respect to ϕ and continuous in t. Let the function $\mathcal{A}_\theta(t, \Phi_{t-k:t}) = \Phi_t + \Delta t$ be a self-attention layer with a residual connection, of context length k and containing learnable parameters $\theta \in R^{d_\theta}$ be a self-attention layer with a residual connection, of context length k and containing learnable parameters $\theta \in R^{d_\theta}$ suppose $A := \{\mathcal{A}_\theta(t,\phi) | \forall \theta \in R^{d_\theta}\}$ be the set of all possible self-attention calculations. Let $\mathcal{M}_i(t, \Phi_{(i-1)\Delta t:t}) = \Phi_t + \Delta t$ be the i-th order explicit Adams method time-integrator. The set of functions up to k-th order $M := \{\mathcal{M}_i | 1 \leq i \leq k\}$ is a subset of A such that $\exists A_i \in A$ s.t. $\|\mathcal{M}_i - \mathcal{A}_{\theta_i}\|_\infty < O(\varepsilon)$, for any $M_i \in M$ and $\varepsilon > 0$. According to their analysis, the linear combination of weighted features from past time is common with the time integration method and it's more suited than deep learning approaches. Additionally, a single transformer layer is significantly more expressive than any numerical solver by learning more complex temporal dependencies.

3.3 Evaluation Methods

Effectively evaluating embedding models for time series prediction depends on their ability to capture meaningful representations. To assess these models we employ a two-phase approach. Firstly, we examine how each model transforms the dataset into a higher-dimensional space. Subsequently, we evaluate their performance on downstream tasks, analyzing how the quality of the initial representation influences overall prediction accuracy.

3.3.1 Data Representation

In the data representation we will evaluate the model based on the construction loss of the embedding model since our autoencoder is trained with multiple loss functions we will evaluate it based on :

- **Reconstruction Error:** When training the Koopman autoencoder, the model learns to reconstruct input data by minimizing the Mean Square Error (MSE) between the real data point and its reconstructed counterpart. This error acts as a measure of how well the model's encoded representation captures the essential information of the data.

- **Prediction Error:** This will assess the model's predictive capabilities by calculating the Mean Squared Error (MSE) between the next input data point and the point predicted using the embedded data and the Koopman operator.

Given that the transformation doesn't guarantee a perfect representation of the data, and the attention mechanism should capture the underlying dynamics, we will prioritise the evaluation of the reconstruction error.

3.3.2 Isotropic Measurement

research studies have studied the importance of analyzing the spatial distribution of word embedding utilized by the first version of GPT and the BERT model doesn't utilize all dimensions of the vector space the occupies which will result in lower predictive ability [6], the embedding that generates the best outcome should inhibit the property of isotropy. A distribution is isotropic when the variance is uniformly distributed across all dimensions and a fully isotropic distribution is when the covariance matrix is proportional to the identity matrix. In contrast, an anisotropic distribution of data is one where the variance is dominated by one or few dimensions. Isotropy has several properties such as mean agnostic means that the measurement of isotropy should not be affected by changes in the mean of the distribution, and maximum variance where if one dimension had an increase in variance while others remain the same the isotropy score should decrease. Rotation invariance requires that the measure of spatial utilization remains unaffected by rotation, as the distribution of principal components doesn't change under rotation. Additionally, isotropy scores should remain consistent when the covariance matrix of the underlying data distribution is multiplied by a positive scalar. Multiple measures we utilized in this study such as:

1. **Average Random Cosine Similarity (ARCS):** is a measure that calculates the average cosine similarity between randomly selected pairs of embeddings calculated by $ARCS = \frac{1}{N}\sum_{i=1}^{N}\cos(v_i, v_j)$. Given that our repressilator model is designed to achieve an oscillatory state, the embedding space should exhibit a slightly higher ARCS to ensure it can effectively capture the intricate relationships and patterns present in the data, but not too high in a way that the GPT model wouldn't differentiate between data points.

2. **partition score:** defined by [33] is the particular quotient involving the partition function $Z(c) := \sum_{x\in X}\exp(c^T x)$ where c is chosen from the eigenspectrum of

XX^T. This score indicates that the space is isotropic if it's closer to 1 and anisotropic if it's closer to 0.

3. **Variance Explained Ratio:** This score function, which we call the VarEx Score, quantifies the extent to which each of the k primary components of the data accounts for the overall variance. A high VarEx score shows that the selected principal component captures the majority of the variability in the data, which is not the case in isotropic embedding space, where variance is uniformly distributed across all dimensions. A low VarEx score is necessary in this scenario, as the first main component does not account for the majority of the variance.

   In terms of how these metrics assess the isotropy of the embedding space, Average Random Cosine Similarity is the most effective at measuring rotation invariance and the proximity of data points within the embedding space, which impacts data reconstruction. The partition Score is most suitable for evaluating the maximum variance in the embedding space, while the Variance Explained (VarEx) Score accounts for mean agnosticism, scalar covariance, and rotation invariance.

3.3.3 GPT model evaluation

To evaluate the GPT model trained on our embeddings, we will use four metrics:

1. Training Error: This metric is the training Mean Squared Error (MSE) between the input embedding and the output embedding. It indicates the GPT model's ability to fit the embeddings.

2. Evaluation Embedding: This metric measures the difference between the input embedding and the output embedding. With a context of 32 tokens and an output of 256 tokens (including the forecast), this allows us to assess the embedding's effectiveness for forecasting purposes.

3. State Error: This error is calculated between the original data and the embedding model. It helps determine how well both the embedding and the GPT model can forecast the real data. In dynamical systems, even minor changes in input dimensions could lead to significantly different outcomes, as seen in systems like the Lorenz system.

Chapter 4

Experiments and Results

4.1 GPT and Embedding Models

The Embedding Model Training Process is as follows:

1. **Dataset Preparation:** The training dataset consists of 2048 trajectories each having 1024 time-steps generated from unique initial conditions for the represillator model. Data is processed with a batch size of 256, each batch contains 16 consecutive data points with a stride of 16 and is shuffled for robust training. A validation dataset of 64 trajectories is generated under the same conditions but with the same batch size of 64 each having 32 consecutive data points with a stride of 32 for efficient validation loss calculation.

2. **Model Training:** The embedding model is trained for 300 epochs, with validation performed every 5 epochs. Model checkpoints (including weights, optimizer, and scheduler) are saved every 25 epochs. The mean and standard deviation for each dimension are calculated from the training data and stored within the model. Data is normalized before input to the embedding network and denormalized after the recovery network and the loss is calculated from real data instead of normalized data.

3. **Optimization:** The model is optimized using the Adam optimizer with a weight decay of 1e-8. The learning rate (1e-3) is adjusted with an exponential scheduler using a gamma of 0.995. To prioritize reconstruction loss, it is multiplied by 1e3, while decay loss is multiplied by 0.1. Prediction loss remains unchanged.

The GPT model training process is as follows:

1. **Dataset Preparation:** The GPT model utilizes the same training and validation dataset as the embedding model. For the training dataset, the input is a sequence of 32 data points used for next-point prediction using the GPT model. As for the validation dataset the sequence is constructed from a sequence of 256 data points, the GPT model uses 32 context lengths and the loss is calculated from the forecasting of the rest of the sequence. In order to accelerate training, the entire training dataset is pre-embedded using the embedding model. For validation, the sequence of real data is normalized and embedded before GPT forecasting and then reconstructed and denormalized using the embedding recovery network to evaluate the performance on real data.

2. **Model Architecture:** The GPT model comprises 4 decoder layers with 4 attention heads each. The GELU activation function is employed due to its smoother, more effective learning capabilities compared to ReLU.

3. **Training Process:** The GPT model is trained for 200 epochs with checkpoints saved every 25 epochs. The Adam optimizer is used, similar to the embedding model, but with a cosine annealing learning rate scheduler found to be more suitable for transformer architectures. After hyperparameter tuning within the range of 1e-3 to 1e-4, a learning rate of 2e-4 was selected for all embedding models.

4.1.1 MLP autoencoder Model

The baseline model uses only Multilayer Perceptron (MLP) that encodes six-dimensional input data into a 32-dimensional embedding space. To determine the optimal architecture, multiple MLP configurations with varying hidden layer sizes (100, 200, and 500 neurons) were explored. A hidden layer size of 500 neurons demonstrated the best performance. Layer normalization is applied prior to the Koopman operator to standardize input data, enhance convergence, and crucially, to generate a more meaningful and informative representation for the Koopman operator to learn from.

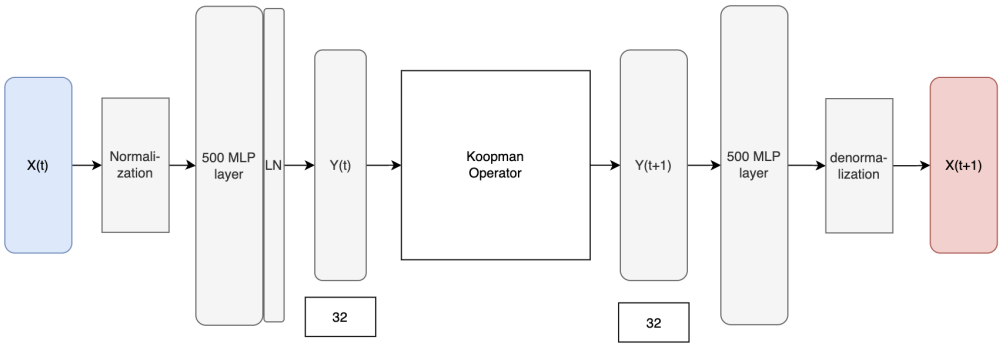


Figure 4.1: Architecture of Koopaman autoencoder using MLP network

The validation dataset is used to build the embedding model results. The average cosine similarity is calculated by taking the average of 100,000 data points. The repressilator model results are as follows:

Metric	**Value**
Number of embedding parameters	39,195
Final Reconstruction Loss	0.00188
Final Prediction Loss	0.06942
Average Cosine Similarity Score	0.88955
Partition Score	0.85792
Varex Score	0.79249

Table 4.1: Model Evaluation Metrics

In comparison to the original data, the average cosine similarity was 0.63209, the embedding model has a higher cosine similarity score, this could be considered an enhancement which indicates that the model effectively captured and accentuated the inherent similarity between the data points. The model also achieves a high partition score reflecting its ability to account for the maximum variance and global stability, the VarEx score analysis revealed that 80% of the variance in the data could be explained by only the first three principal components which might suggest a potential factor is that the model achieve good compression for the data but also it might affect losing nuanced detail that should be spread out to more than the dominant components.

The results after training the GPT model using this architecture for Koopman Embedding are as follows:

The results indicate that after 256-time steps, the state error, defined as the mean

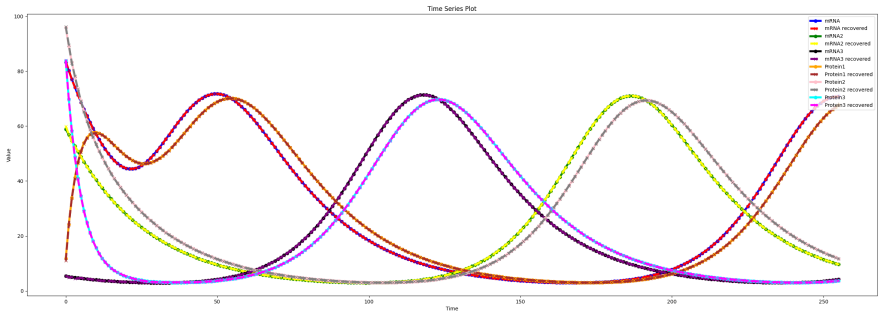


Figure 4.2: sequence of 256 data points from real data and reconstructed data using MLP autoencoder model

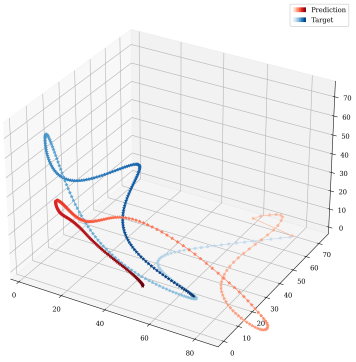
Metric	**Value**
Final Training Error	7.96987×10^{-7}
Final Validation Error	0.00463
State Error	40.195

Table 4.2: Summary of GPT Errors

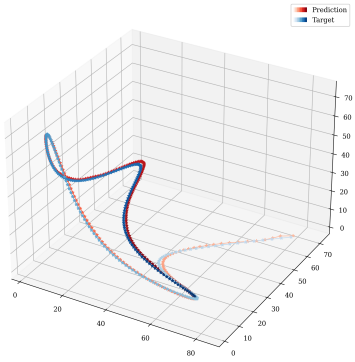
square error of the discrepancies in data points recovered from the GPT's predictions, is 40. This error can be attributed to several factors. Firstly, the GPT model did not achieve the optimal validation error necessary for generating accurate forecasts, especially when compared to the validation error during training, which was conducted over only 32-time steps as opposed to 256. Additionally, the reconstruction loss and cosine similarity score suggest that even small errors in these metrics can lead to significantly different outcomes. The reconstruction loss depicted in Figure 4.2 demonstrates the ability of our model to accurately reconstruct 256-timestep time series from the validation data. For a detailed comparison of reconstruction loss across different Koopman embedding architectures, please refer to the Appendix.. Despite these challenges, the figure below demonstrates that the GPT model effectively captured the dynamics of the system using multivariate data embedded with Koopman embedding, Figure 4.3 shows the difference in the learning of the embedding from the 1st epoch and epoch 200.

To understand the need for a decoder-only transformer on top of the Koopman framework, it is essential to understand the limitation of the Koopman operator for long-horizon predictions. while Koopman's theory suggests that in order to get to $y(n)$ to apply the Koopman operator n times from the initial condition, Figure 4.4 illustrates the state error per time step when using the Koopman operator and our embedding.

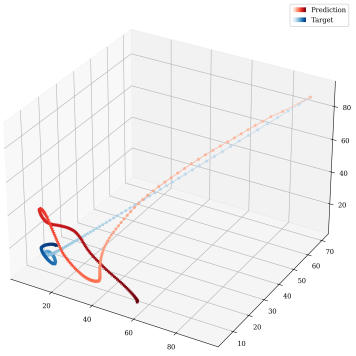
The error arises from several factors. First, the linearization approximation is often accurate for single time-step prediction, particularly when the time step is small and the system's state remains close to its current trajectory. However, this approximation is less accurate for multi-timestep prediction, where the linearization error accumulates and minor nonlinearities might become significant when compounded over several steps. Additionally, the assumption of invariant subspace might hold in short-term prediction and often fails in longer prediction as the system evolves it may explore regions of the state space that are not captured by selected observables. Finally, as noted in [26], the presence of a continuous spectrum is common in many physical systems where it's not straightforward to finitely approximate in terms of a small set of eigenfunctions further challenges the effectiveness of the Koopman operator for long-term predictions.



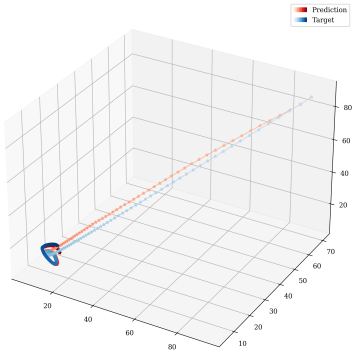
[epoch 0]



[epoch 200]



[epoch 0]



[epoch 200]

Figure 4.3: Two examples of the transform model's progression that show the mRNA level as a three-dimensional plot

4.2 KAN Models

In the KAN architecture, we designed two architectures to leverage the capabilities of the KAN network, as shown in Figure 4.5 and Figure 4.6. The first architecture

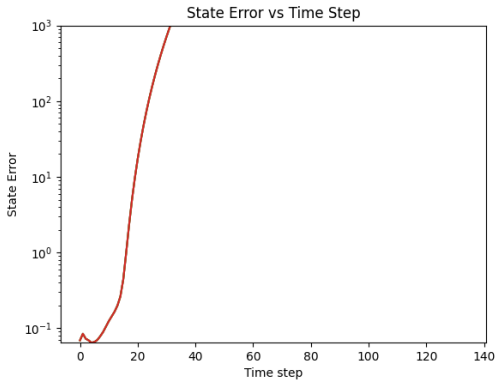


Figure 4.4: State Error per time steps for Koopman operator

uses the KAN network in both the encoder and the decoder. Since the KAN network requires the input to be specified in the grid range for spline function increasing the grid would also require increasing the grid size and that would increase the number of parameters exponentially, we want to use the default range of [-1,1] we employed min-max normalization instead of data standardization which scales the data between 0 and 1, before feeding it into KAN layer with 50 neurons. To ensure that the input to the decoder also falls within the same range we added a Tanh activation after the output of the Koopman operator output, which transform the input into the [-1,1] range before passing it to the KAN network.

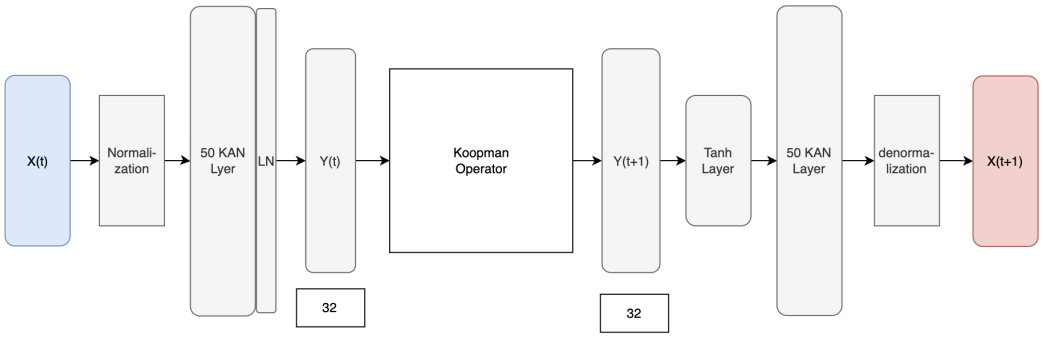


Figure 4.5: KAN network for Koopman embedding

In the second architecture, shown in Figure 4.6, we utilized a mixture of MLP and KAN network to use the original data standardization technique, where it's considered more suitable for autoencoder architecture. This is because autoencoders often handle complex data distribution, where outliers can significantly impact the performance and it's more suitable because many activation functions, such as ReLU, assume a zero-centered input distribution, which is achieved through standardization. To ensure

that the data remains within the standard range required by the KAN network in both the encoder and decoder, we introduced a small MLP layer with the same dimensionality as the input. In the encoder, we employed a 6-neuron MLP layer followed by a Tanh layer. Similarly, in the decoder, we used a 32-neuron MLP layer and a Tanh layer to adjust the output from the embedding and the Koopman operator into the necessary range for the KAN network.

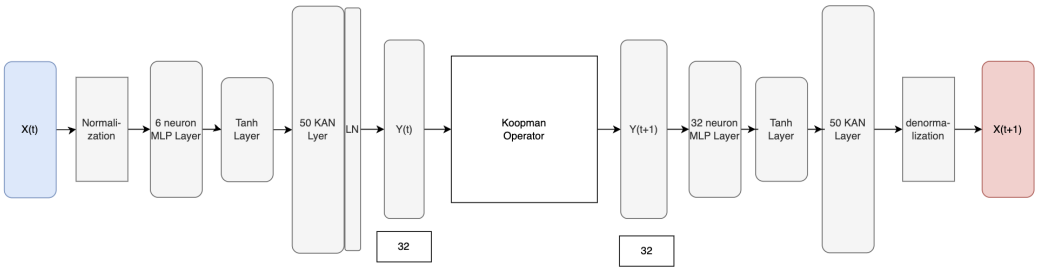
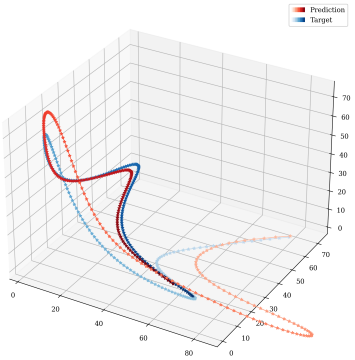
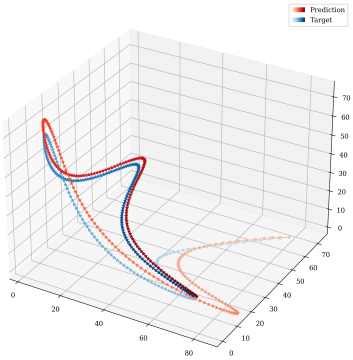


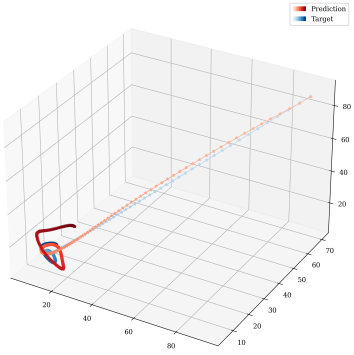
Figure 4.6: KAN network for Koopman with MLP input transformation



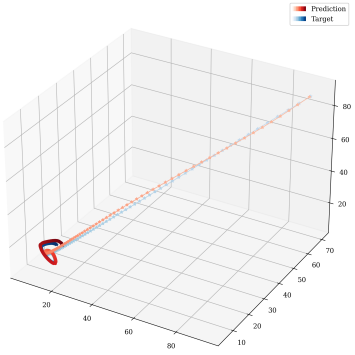
[KAN]



[MLP+KAN]



[KAN]



[MLP+KAN]

Figure 4.7: Comparing the performance of GPT models trained on KAN and KAN+MLP embeddings

Metric	KAN	MLP+KAN
Number of embedding parameters	39,195	39,255
Final Reconstruction Loss	0.00075	0.00160
Final Prediction Loss	0.31980	0.23536
Average cosine similarity score	0.97899	0.90267
Partition score	0.87915	0.88969
Varex Score (Variance Explained)	0.90605	0.91733
GPT Final Training Error	1.30914e-07	5.07036e-07
GPT Final Validation Error	0.01362	0.00718
GPT State Error	283.800	54.619

Table 4.3: Comparison of Metrics for KAN and MLP+KAN

The results for the embedding models applied to the repressilator model are presented in Table 4.3 . These results highlight the influence of the isotropic measure and cosine similarity score on the ability of the GPT model to learn from Koopman embeddings. Although the KAN model achieved the best reconstruction error, with a value of 7e-4, which is lower than the MLP model's error of 0.0018 and the MLP+KAN model's error of 0.0016. The GPT results demonstrate the impact of improved dynamics learning in Koopman by examining the prediction loss and the cosine similarity score, although a higher cosine similarity score is expected, an excessively high score can impair the GPT ability to distinguish between data points, leading to increased validation error and thus a higher state error. To illustrate this, figure 4.7 shows that even though the embedding model captures the overall dynamics, visualizing the data on a real scale reveals that there is a significant discrepancy between the predicted and actual data points.

4.3 Differnet Arch for Encoder and Decoder

In the following two experiments, we investigate the impact of using different architectures for the encoder and decoder. In the first architecture shown in Figures 4.8, we employed a KAN network in the encoder and MLP in the decoder. We experimented with various configurations for the encoder by adjusting the learning rates, batch sizes, and embedding dimensions. After extensive testing, we found out the best result could be achieved by only increasing the embedding dimension to 64 from 32 and expanding

the grid range from [-1,1] to [-3,3] to ensure all normalized data falls within this region for B-spline, that increased the model's ability to take a bigger range of the input without the need to increase the grid size. Despite these adjustments, the results did not meet the performance of the previous architectures. To address this, we introduced a layer normalization step before feeding the input to the next KAN network, standardizing the input for each KAN. This modification led to reconstruction and prediction losses that were more in line with the results of previous experiments. The addition of layer normalization is particularly useful for addressing the covariate shift and ensuring the input remains within the specified range, this strategy has been validated recently by [2] and [21].

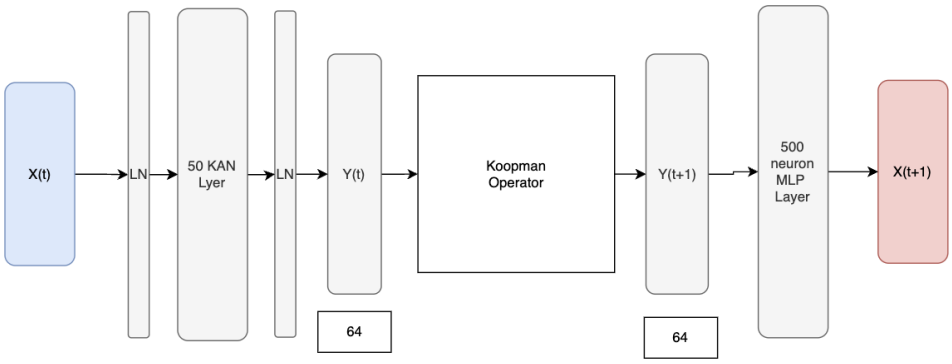


Figure 4.8: KAN encoder - MLP decoder

The second architecture in Figure 4.9 reverses the roles of the KAN and MLP networks. using an MLP in the encoder and a KAN network, using an MLP in the encoder and a KAN network in the decoder part. This allows us to examine how the effectiveness of different architectures is most effective at finding the optimal embedding dimension and which is better suited for reconstructing the output from the embedded dimensions.

The result from the embedding and GPT training shows that the autoencoder with an MLP encoder outperforms the one with a KAN encoder in both the prediction and reconstruction loss. This is consistent with the initial model, where the MLP network achieved better prediction loss. These findings indicate that the MLP network is more effective at identifying Koopman embedding that is closer to linear observables compared to KAN networks.

Moreover, the MLP encoder produced a better partition score than the KAN encoder which indicates a clearer separation in the embedding space. This aligns with the

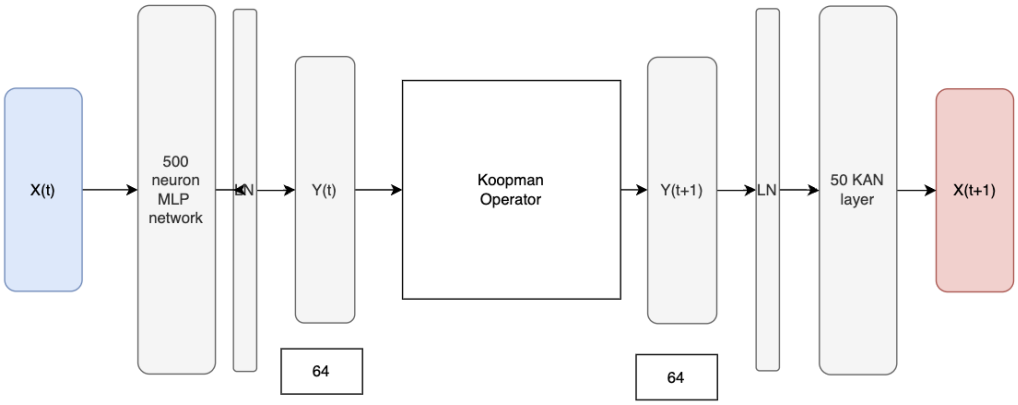


Figure 4.9: MLP encoder - KAN decoder

cosine similarity score where the KAN encoder performed similarly to the original data, suggesting it did not learn the embedding as effectively. The autoencoder loss metrics further emphasize how the GPT model's performance is influenced by the choice of embedding model. The higher validation error and reconstruction loss with the KAN encoder suggest that it struggles to capture the system's dynamics, leading to a higher state error in the GPT model. This underscores the importance of selecting the right encoder architecture to optimize both embedding quality and downstream performance.

Metric	**Enc KAN Dec MLP**	**Enc MLP Dec KAN**
Final Reconstruction Loss	0.00484	0.00126
Final Prediction Loss	0.39489	0.15759
Average Cosine Similarity Score	0.68733	0.82566
Partition Score	4.04926e-11	0.00978
Varex Score (Variance Explained)	0.74972	0.87193
Final GPT Training Error	2.97040e-06	2.11200e-06
Final GPT Validation Error	0.03579	0.00432
Final GPT State Error	135.5809	12.51438

Table 4.4: Comparison of Metrics for Encoder-Decoder Architectures

The graphs in Figure 4.11 illustrate the loss metrics for both reconstruction and prediction when using the Koopman autoencoder, based on various architectural configurations we tested. In contrast, Figure 4.12 presents the results of training these

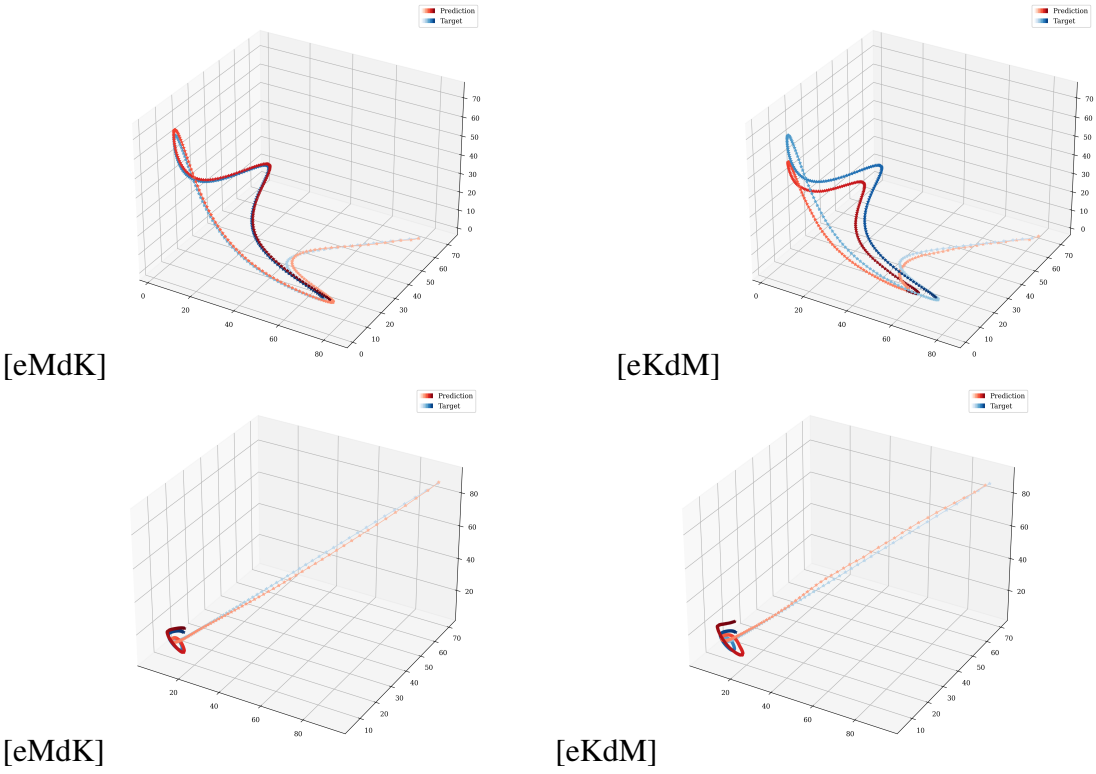


Figure 4.10: Comparing the performance of GPT models trained on ”KAN encoder and MLP decoder ” and ”MLP encoder and KAN decoder” embeddings

different architectural choices on dynamic systems using a GPT model, highlighting how each architecture influences the model's performance.

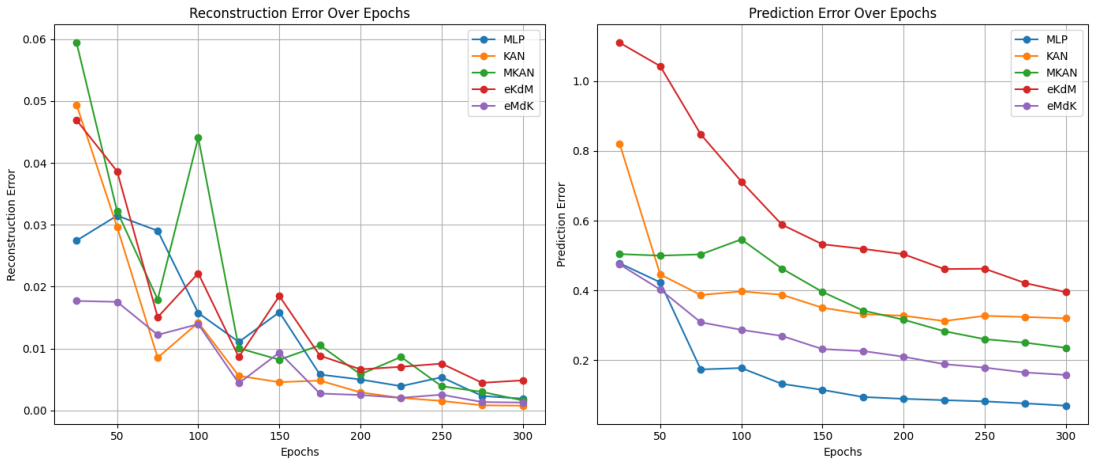


Figure 4.11: Reconstruction and Prediction Error for different Koopman embedding Structures

To evaluate whether our model effectively preserves the structure of the original data, we employed UMAP (Uniform Manifold Approximation and Projection). UMAP

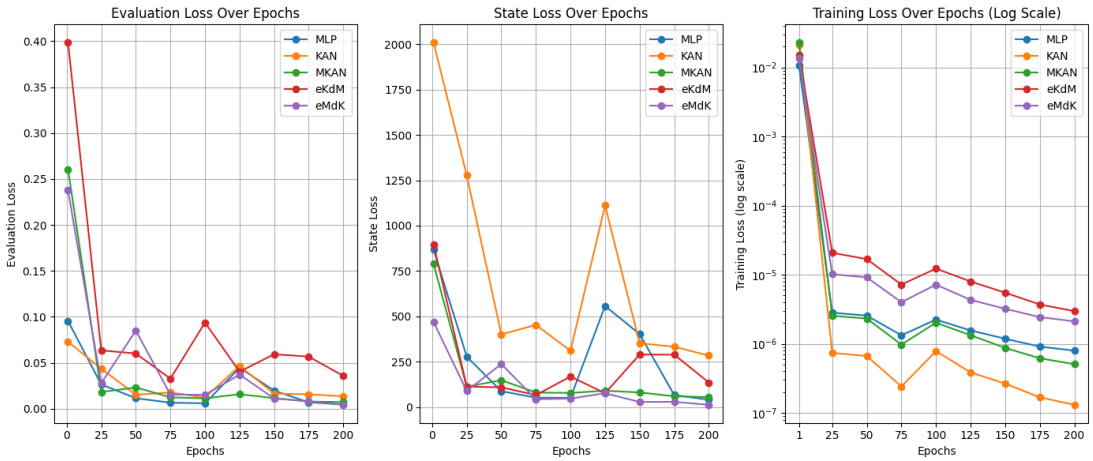


Figure 4.12: GPT training results using different Embedding Structures

is a dimensionality reduction technique that projects high-dimensional data into a lower-dimensional space while preserving both local and global structures of the data. By applying UMAP, we can visualize the relationships between data points in the original high-dimensional space and compare them directly to the relationships in the embedded space produced by our models. This comparison allows us to assess whether the essential patterns, clusters, and neighbourhood relationships are maintained, providing insights into the model's ability to retain the integrity of the original data structure during the embedding process. For a quantitative measure, we also utilize the Jaccard Index score where a higher value indicates that the embedding preserves the neighbourhood structure of the original data more faithfully, while a higher Hit Rate suggest that more of the nearest neighbours in the original space are retained in the embedding space.

From Figure 4.13 and Table 4.5 all embeddings can effectively preserve the overall data structure, as indicated by their high Jaccard Index scores and hit rates. The KAN network demonstrates the best performance among the models, achieving the highest values for both metrics, followed by the combination of MLP and KAN, which also shows strong data structure preservation. In contrast, the lowest performance is observed when using KAN as the encoder and MLP as the decoder, indicating less effective preservation of the data structure in this configuration. This preservation of data structure explains the enhanced ability of the GPT model to learn from the training data, as illustrated in Figure 4.12. The performance follows the same ranking as observed in the Jaccard Index and Hit Rate scores.

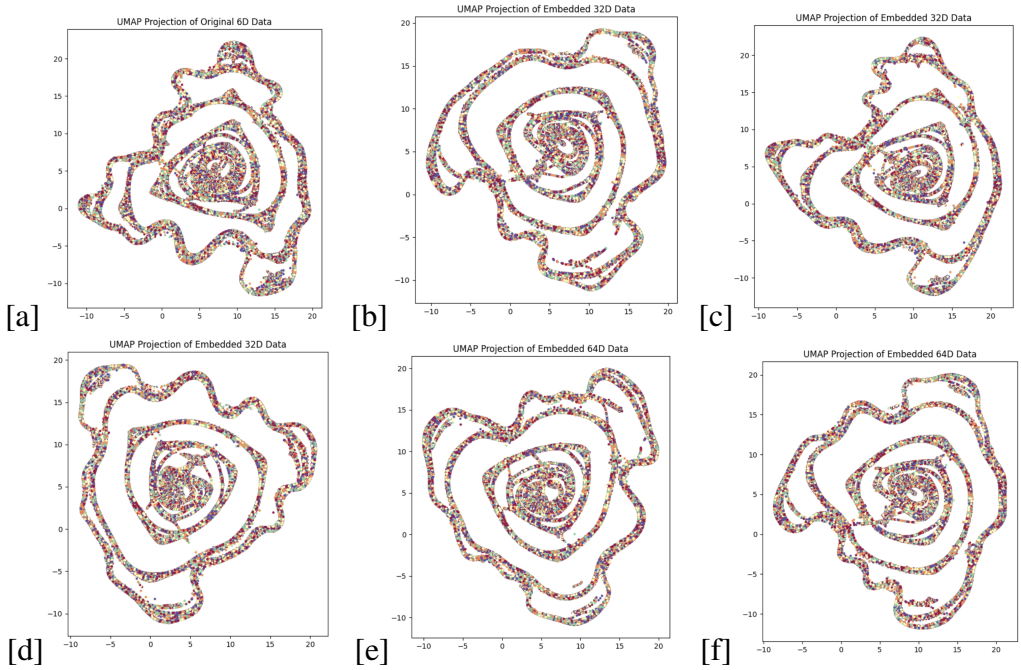


Figure 4.13: UMAP for data in different embeddings. [a] validation data, [b] MLP embeddings, [c] KAN embedding, [d] mKAN embeddings, [e] eKdM embeddings, [f] eMdK embeddings

Model	Mean Jaccard Index	Hit Rate
MLP	0.7554	0.8509
KAN	0.9024	0.9446
mKAN	0.8038	0.8837
eKdM	0.7353	0.8346
eMdK	0.7523	0.8489

Table 4.5: Comparison of Mean Jaccard Index and Hit Rate for Different Models

4.4 Foundational Models

TO evaluate the performance of our model, we will compare it against a foundational model in time series forecasting. The state-of-the-art models are Chronons [3], MOIRAI [43], TimesFM [10], and MOMENT [15], each employs distinct architectures and approaches for forecasting. The Chronons model leverages the T5 architecture to learn the underlying patterns and relationships, they use a mix of synthetic data (TSMixup and KernelSynth) and real-world datasets. Similarly, TimeFM uses a mixer of synthetic datasets, Google Trends and other publicly available datasets, they utilize a decoder-

only model with input patching. In contrast, MOMENT gathered a large time-series corpus called data pile and trained an encoder-only model with a forecasting head and classification head, this model is trained on multiple tasks such as forecasting, anomaly detection, and classification. MOIRAI model uniquely handles multivariate data by employing patched input, variate IDs, and time IDs to embed multivariate time series. For our comparison, we selected the MOMENT model due to its superiority in dealing with multivariate data by embedding each channel separately while sharing parameters across channels, thereby enhancing forecasting accuracy. Among other models, only MOIRAI can handle multivariate data but relies on a fixed approach, requiring specific historical data parameters and a particular data loader architecture.

We provided a context of 32, as in the GPT model, and padded the input vector with zero as recommended in the article to provide a more fair comparison between the GPT and MOMENT models. The following are the outcomes after fine-tuning MOMENT for 10 epochs when compared to our best GPT using Koopman embedding:

Model	Validation Embedding Error	State MSE Error
GPT Model	0.0043	12.51
MOMENT Model	0.402	123.284

Table 4.6: Comparison of Validation Embedding Error and State MSE Error between GPT and MOMENT models.

The prediction for each channel in MOMENT, as illustrated in Figure 4.14 shows that MOMENT is capable of capturing the general trend within individual channels. However, it lacks the capability to capture interdependence between channels. As a result, the model's predictive power may be constrained when dealing with complex multivariate data where cross-channel interaction plays a significant role in forecasting accuracy.

4.5 Overview of Current KAN Implementation and Activation Function Extraction for Koopman

There are several implementations of KAN networks with the most prominent being: the original implementation by [24] available in the pyKAN repository, a more efficient implementation are provided by the efficientKAN repository, and a third variant is

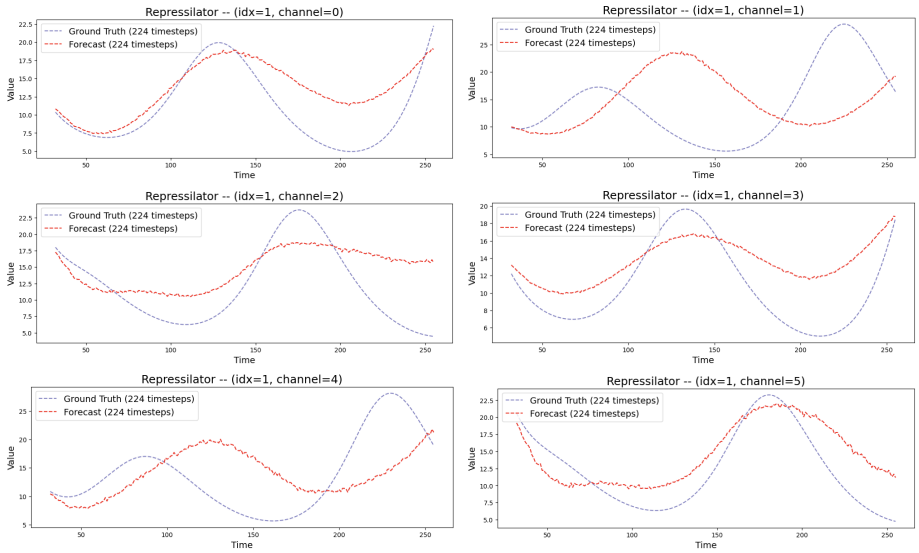


Figure 4.14: MOMENT modelling Repressilator data as channels

fastKAN which replaces B-spline functions with approximation using Radial Basis Functions (RBF).

In our work, we chose to use the efficientKAN implementation due to the significant limitation of the original implementation. Firstly, the original implementation requires expanding the input tensor to a shape of (batch size, out Features, in Features) for applying each activation function which increases the computation time. Secondly, it only accepts input in the form [batch size, dimension], whereas, for time series tasks and our Koopman implementation, the input is typically in shape [batch size, number of time-steps, dimensions]. Lastly, the original implementation is more challenging to integrate with MLP architectures. The efficentKAN implementation addresses these issues by reformulating the computation where it weights the results of all B-spline functions for all nodes to produce output directly which in turn enhances the efficiency. However, the downside is that their way of computation makes it harder to compute the spline weights for the node[i][j] this results in a way that it's harder to prune the network and plot the activation functions.

To illustrate how KAN be utilized to deterministically determine the activation function, we implemented the Koopman embedding training process as described in [26]. In this approach, the Koopman operator is parameterized by an auxiliary network which is used to dynamically compute different operators based on the value of the observables, effectively addressing the challenges of continuous spectrum and limited invariant subspace discussed in the previous section. In our implementation, we employed 16 observables calculated by the Koopman autoencoder. The auxiliary

network then produces a set of 8 complex conjugate eigenvalues $\lambda = \mu \pm i\omega$ which are subsequently used to construct the Koopman operator by:

$$B(\mu,\omega) = \exp(\mu\Delta t)\begin{bmatrix}\cos(\omega\Delta t) & -\sin(\omega\Delta t)\\ \sin(\omega\Delta t) & \cos(\omega\Delta t)\end{bmatrix}$$

This method allows for a more flexible and accurate representation of the system's dynamic, enabling the network to dynamically adjust to different regions of the subspace. since efficientKAN couldn't be used to plot the activation at the moment, we opted to use fastKAN in the process where it's used in the encoder, the decoder, and the auxiliary network.

The following plot shows the effect of the first four embedding values on the calculation of μ_1 and ω_1, where $\phi_{p,q}$ refer to the activation function between the input p and the output q.

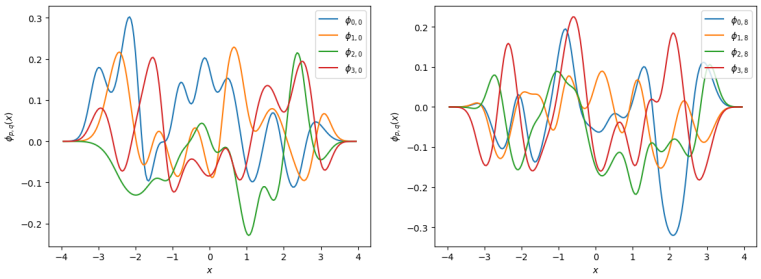


Figure 4.15: The effect of different input ranges on eignvalues

4.6 Discussion

Based on the results obtained from training Koopman embedding models and GPT on the repressilator dataset, it's essential to consider isotropic measures, as they impact the later stage of the GPT model validation and their ability to forecast the dynamics on unseen data. Metrics such as cosine similarity, partition score, and explained variance should be thoroughly analyzed beforehand to ensure effective model performance. for improved forecasting on real data, the Koopman embedding should exhibit low error for prediction error that would reflect how well our embedding can capture the necessary observable and dynamics to reinforce physical context. The reconstruction error is particularly critical in training the GPT model, as it directly influences the ability to accurately reconstruct real data.

Preserving the original data structure is another factor that influences the easier training of the GPT model. Using Quantitative measures like the Jaccard Index and Hit Rate are strong indicators of how well the embedding preserves the original data structure, and using UMAP for visualizing data structures under different embedding models could provide valuable insights.

When training the GPT model, data standardization and normalization techniques are critical in enhancing model performance, improving data quality and helping mitigate the effect of outliers. In training LLM for time series forecasting this is useful in capturing the trend, seasonality and nonlinearities for multivariate time series. In dynamical systems, accurately capturing the real values is vital as information loss is one of the side effects of data standardization, as small errors can lead to significant deviation in dynamics, as seen in systems such as the Lorenz system.

For training a GPT model that needs to correlate with the entire dynamics of systems like the repressilator model, the Koopman embedding should be trained on parameters that are unstable and capture oscillatory behaviour to encompass a wide range of dynamics, with isotropy enhancement architecture in mind. When only training a Koopman embedding model without a later stage of GPT, it's important to use a parametrized Koopman operator, as fixed Koopman operators can accumulate significant errors over multiple time steps. Finally, when using a KAN network, careful consideration of the autoencoder's architectural design is crucial, ensuring that the network appropriately handles the required data and layer propagation.

Chapter 5

Conclusion

The development of system identification for biological and physical systems has evolved from using optimization techniques to fit existing mathematical models to data-driven approaches like Dynamic Mode Decomposition (DMD), and more recently, machine learning and transformer models. This work provided a way to bridge the gap between models traditionally used for physical systems and their application to biological data, specifically utilizing the repressilator model. Key aspects of biological data include achieving high prediction accuracy and constructing interpretable systems. To address this, we utilized techniques using Koopman embeddings and GPT models to learn the dynamics of the repressilator, exploring various architectures that incorporate MLP and KAN networks.

Our work extends the application of KAN beyond its original scope, demonstrating its potential for use in larger systems. KAN network offers the ability to prune and visualize univariate functions between nodes, which can eventually help capture the full mathematical model of the system. Building a Koopman embedding model with KAN requires careful architectural design and a deep understanding of KAN's learning requirements.

This research builds on previous system identification studies by extending architectural variations that can be utilized and improved. Koopman embedding might work for its original purpose, but it might fail at a later stage of the training pipeline, emphasizing the need to carefully analyze embedding metrics. Future research should expand the applicability of our model to a wider variety of biological systems and build an embedding that could be used for any system identification. Finally, further development should take into account the different implementations of the KAN network where it's easy to prune and extract the mathematical framework.

Bibliography

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Mohammed Ghaith Altarabichi. Rethinking the function of neurons in kans. *arXiv preprint arXiv:2407.20667*, 2024.

[3] Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, et al. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*, 2024.

[4] Steven L Brunton, Marko Budišić, Eurika Kaiser, and J Nathan Kutz. Modern koopman theory for dynamical systems. *arXiv preprint arXiv:2102.12086*, 2021.

[5] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[6] Xingyu Cai, Jiaji Huang, Yuchen Bian, and Kenneth Church. Isotropy in the contextual embedding space: Clusters and manifolds. In *International conference on learning representations*, 2021.

[7] Defu Cao, Furong Jia, Sercan O Arik, Tomas Pfister, Yixiang Zheng, Wen Ye, and Yan Liu. Tempo: Prompt-based generative pre-trained transformer for time series forecasting. *arXiv preprint arXiv:2310.04948*, 2023.

[8] Minjong Cheon. Kolmogorov-arnold network for satellite image classification in remote sensing. *arXiv preprint arXiv:2406.00600*, 2024.

[9] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al.

Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6, 2023.

[10] Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. *arXiv preprint arXiv:2310.10688*, 2023.

[11] Michael B Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[12] Marco Forgione, Filippo Pura, and Dario Piga. In-context learning for model-free system identification. *arXiv preprint arXiv:2308.13380*, 2023.

[13] Nicholas Geneva and Nicholas Zabaras. Transformers for modeling physical systems. *Neural Networks*, 146:272–289, 2022.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[15] Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. Moment: A family of open time-series foundation models. *arXiv preprint arXiv:2402.03885*, 2024.

[16] Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, et al. Time-llm: Time series forecasting by reprogramming large language models. *arXiv preprint arXiv:2310.01728*, 2023.

[17] Ming Jin, Qingsong Wen, Yuxuan Liang, Chaoli Zhang, Siqiao Xue, Xue Wang, James Zhang, Yi Wang, Haifeng Chen, Xiaoli Li, et al. Large models for time series and spatio-temporal data: A survey and outlook. *arXiv preprint arXiv:2310.10196*, 2023.

[18] Karel J Keesman. *System identification: an introduction*. Springer Science & Business Media, 2011.

[19] Andreĭ Kolmogorov. *On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables*.

[20] Yunzhu Li, Hao He, Jiajun Wu, Dina Katabi, and Antonio Torralba. Learning compositional koopman operators for model-based control. *arXiv preprint arXiv:1910.08264*, 2019.

[21] Ziyao Li. Kolmogorov-arnold networks are radial basis function networks. *arXiv preprint arXiv:2405.06721*, 2024.

[22] Minghao Liu, Shengqi Ren, Siyuan Ma, Jiahui Jiao, Yizhou Chen, Zhiguang Wang, and Wei Song. Gated transformer networks for multivariate time series classification. *arXiv preprint arXiv:2103.14438*, 2021.

[23] Xin Liu, Daniel McDuff, Geza Kovacs, Isaac Galatzer-Levy, Jacob Sunshine, Jiening Zhan, Ming-Zher Poh, Shun Liao, Paolo Di Achille, and Shwetak Patel. Large language models are few-shot health learners. *arXiv preprint arXiv:2305.15525*, 2023.

[24] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024.

[25] Lennart Ljung, Carl Andersson, Koen Tiels, and Thomas B Schön. Deep learning and system identification. *IFAC-PapersOnLine*, 53(2):1175–1181, 2020.

[26] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950, 2018.

[27] Oren Melamud, Jacob Goldberger, and Ido Dagan. context2vec: Learning generic context embedding with bidirectional lstm. In *Proceedings of the 20th SIGNLL conference on computational natural language learning*, pages 51–61, 2016.

[28] Fredric M Menger and Jason S Keiper. Gemini surfactants. *Angewandte Chemie International Edition*, 39(11):1906–1920, 2000.

[29] Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, et al. Crosslingual generalization through multitask finetuning. *arXiv preprint arXiv:2211.01786*, 2022.

[30] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. February 2018.

[31] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[32] Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Arian Khorasani, George Adamopoulos, Rishika Bhagwatkar, Marin Biloš, Hena Ghonia, Nadhir Vincent Hassen, Anderson Schneider, et al. Lag-llama: Towards foundation models for time series forecasting. *arXiv preprint arXiv:2310.08278*, 2023.

[33] William Rudman, Nate Gillman, Taylor Rayne, and Carsten Eickhoff. Isoscore: Measuring the uniformity of embedding space utilization. *arXiv preprint arXiv:2108.07344*, 2021.

[34] Peter J Schmid. Dynamic mode decomposition and its variants. *Annual Review of Fluid Mechanics*, 54(1):225–254, 2022.

[35] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

[36] Subhrajit Sinha, Bowen Huang, and Umesh Vaidya. Robust approximation of koopman operator and prediction in random dynamical systems. In *2018 Annual American Control Conference (ACC)*, pages 5491–5496. IEEE, 2018.

[37] Tina Toni and Michael PH Stumpf. Simulation-based model selection for dynamical systems in systems and population biology. *Bioinformatics*, 26(1):104–110, 2010.

[38] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[39] Shreshth Tuli, Giuliano Casale, and Nicholas R Jennings. Tranad: Deep transformer networks for anomaly detection in multivariate time series data. *arXiv preprint arXiv:2201.07284*, 2022.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[41] 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.

[42] Matthew O Williams, Clarence W Rowley, and Ioannis G Kevrekidis. A kernel-based approach to data-driven koopman spectral analysis. *arXiv preprint arXiv:1411.2260*, 2014.

[43] Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. *arXiv preprint arXiv:2402.02592*, 2024.

[44] Haixu Wu, Huakun Luo, Yuezhou Ma, Jianmin Wang, and Mingsheng Long. Ropinn: Region optimized physics-informed neural networks. *arXiv preprint arXiv:2405.14369*, 2024.

[45] Kunpeng Xu, Lifei Chen, and Shengrui Wang. Kolmogorov-arnold networks for time series: Bridging predictive power and interpretability. *arXiv preprint arXiv:2406.02496*, 2024.

[46] Runpeng Yu, Weihao Yu, and Xinchao Wang. Kan or mlp: A fairer comparison. *arXiv preprint arXiv:2407.16674*, 2024.

[47] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8980–8987, 2022.

[48] Lotfi A Zadeh. From circuit theory to system theory. *Proceedings of the IRE*, 50(5):856–865, 1962.

[49] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.

Appendix A

Reconstruction Loss

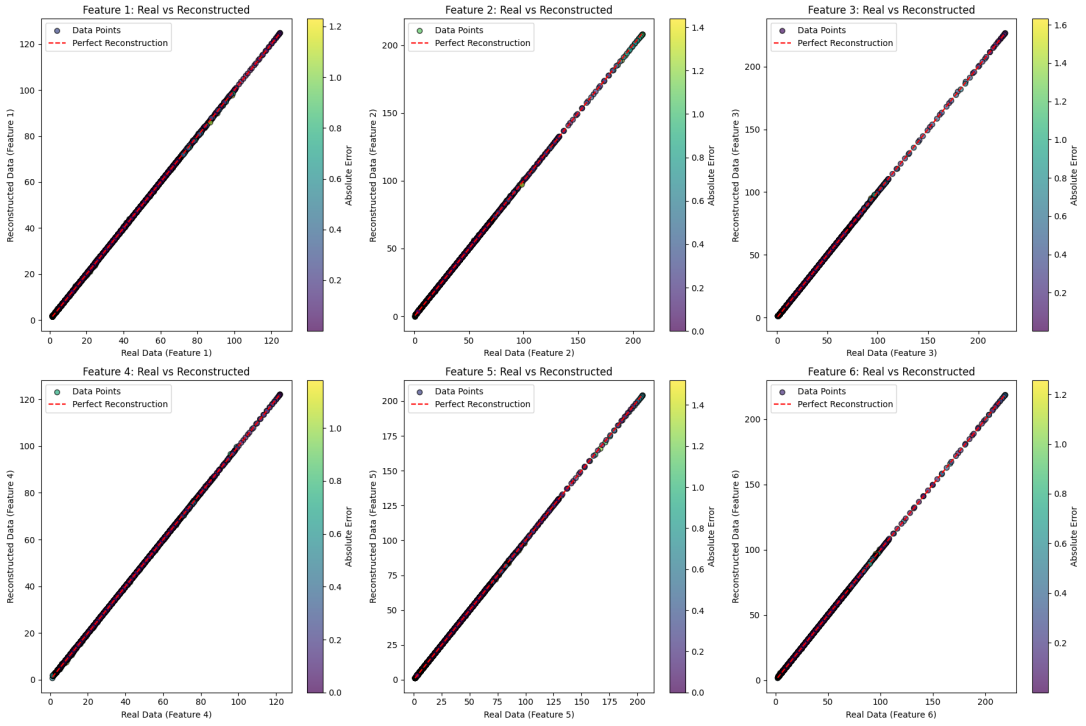


Figure A.1: Reconstruction for MLP Autoencoder

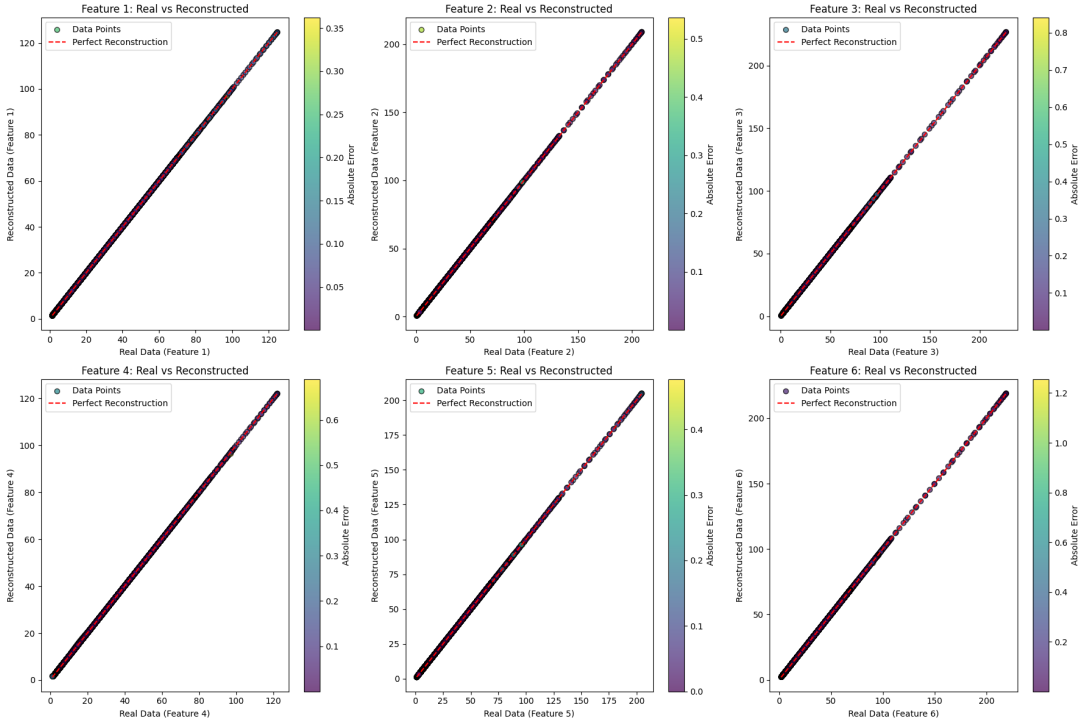


Figure A.2: Reconsturction for KAN Autoencoder

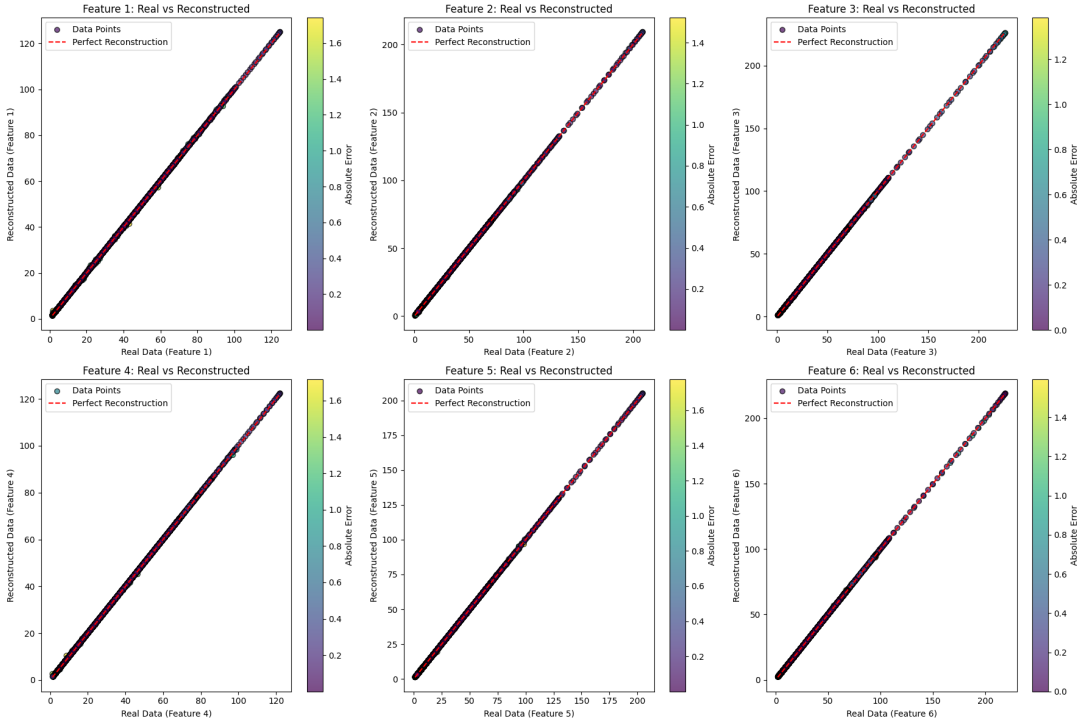


Figure A.3: Reconstruction for MLP+KAN Autoencoder

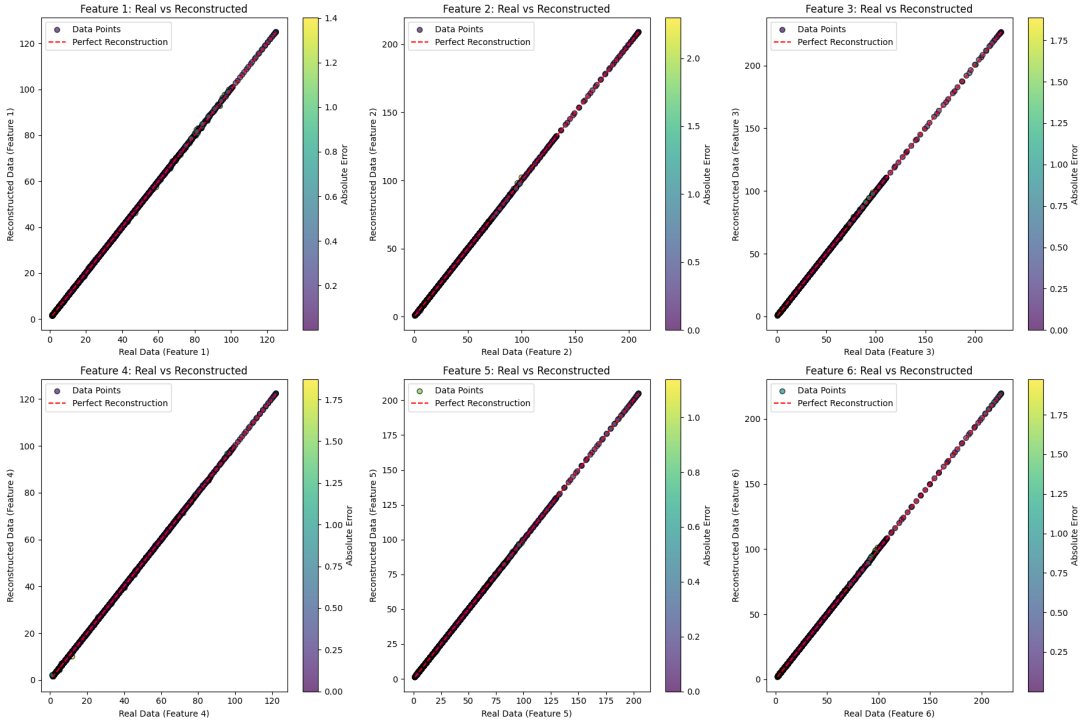


Figure A.4: Reconstruction for MLP Encoder and KAN Decoder

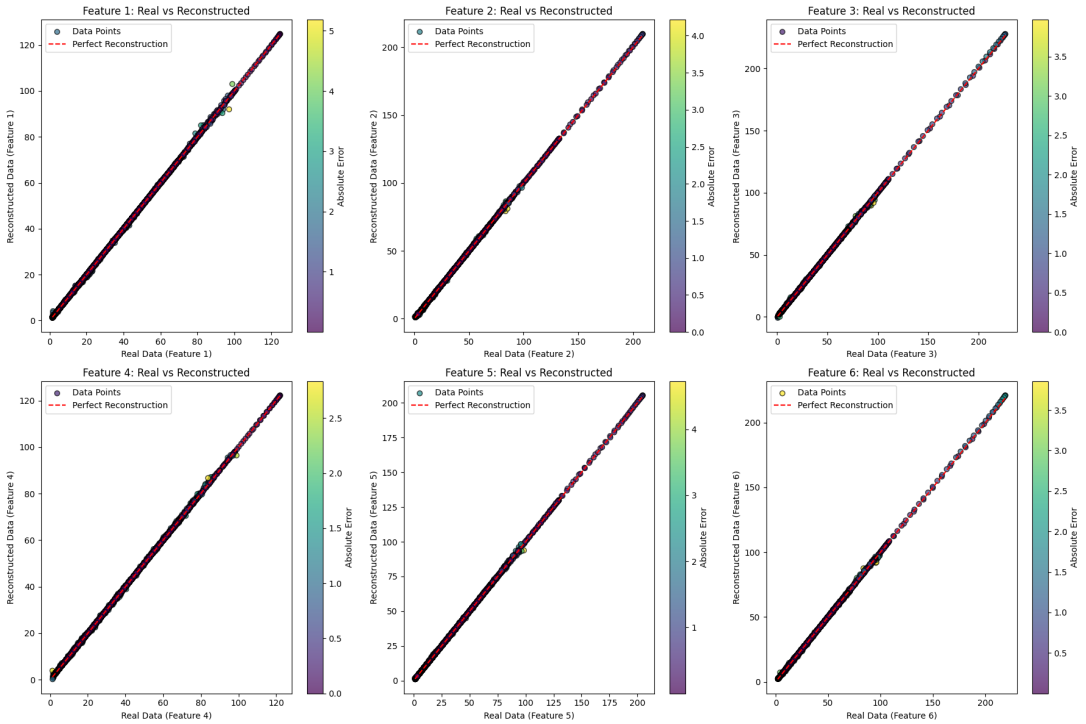


Figure A.5: Reconstruction for KAN Encoder and MLP Decoder