Unveiling LLM Mechanisms Through Neural ODEs and Control Theory

Anonymous ACL submission

Abstract

This paper proposes a framework combining Neural Ordinary Differential Equations (Neural ODEs) and robust control theory to enhance the interpretability and control of large language models (LLMs). By utilizing Neural ODEs to model the dynamic evolution of input-output relationships and introducing control mechanisms to optimize output quality, we demonstrate the effectiveness of this approach across multiple question-answer datasets. Experimental results show that the integration of Neural ODEs and control theory significantly improves output consistency and model interpretability, advancing the development of explainable AI technologies.

1 Introduction

1.1 The Challenge of Interpreting LLMs

Large Language Models (LLMs) have demonstrated impressive performance across a range of natural language processing tasks, from machine translation to text generation and summarization(Brown et al., 2020). Despite their remarkable capabilities,interpreting the decision-making processing of these models remains a significant challenge. The opacity of LLMs raises critical questions about their reliability, fairness, and ethical implication in real-word application (Lipton, 2017). Understanding the underlying mechanisms of LLMs is essential for building trust and accountability in AI systems, particularly when these systems are deployed in high-stakes environments such as healthcare and low.

2 Literature Review

2.1 Current Methods for Enhancing Interpretability in LLMs

The interpretability of large language models (LLMs) has become a central concern in AI research. To address this challenge, various approaches have been proposed, which can broadly be categorized into local and global analyses.

2.1.1 Local Analysis

Local analysis focuses on interpreting individual predictions made by a model by examining specific input-output relationships. The primary goal is to understand the contribution of each input feature to the model's output. Key approaches within local analysis include feature attribution methods and analyzing transformer blocks.

Feature attribution methods quantify the influence of each input feature on the model's predictions. Common techniques include gradient-based methods and vector-based methods. Gradient-based methods, such as Integrated Gradients (Sundararajan et al., 2017), compute the gradients of the output with respect to inputs, attributing significance based on how changes in input features affect the model's predictions. Vector-based methods, such as the Shapley Value framework (Lundberg and Lee, 2017), evaluate the contribution of each feature by considering all possible combinations of input variables. These methods allow researchers to identify influential features and gain insights into why the model produces specific outputs.

For transformer-based models like BERT and GPT, analyzing components such as multi-head self-attention (MHSA) and multi-layer perceptron (MLP) sublayers can provide valuable insights into the model's behavior. For example, examining MHSA sublayers reveals how attention weights are distributed across input tokens, helping to determine if the model focuses on relevant words or phrases (Vig, 2019). Similarly, analyzing MLP sublayers reveals how feature combinations are processed and transformed, elucidating the flow of information through the network. These analyses can uncover how specific tokens drive shifts in attention across layers, leading to richer interpretations of model decisions (Beltagy et al., 2020).

2.1.2 Global Analysis

In contrast to local analysis, global analysis seeks to provide a broader understanding of a model's behavior by exploring the underlying principles that govern its representations and knowledge. This approach includes probing-based methods and mechanistic interpretability.

Probing-based methods involve training auxiliary models on hidden representations to assess the knowledge encoded within the model. Techniques such as probing knowledge and probing representations allow researchers to evaluate the extent to which LLMs capture linguistic properties, syntactic structures, or semantic features (Tenney et al., 2019). These methods help identify the high-level knowledge embedded in the model, offering insights into its internal decision-making processes.

Mechanistic interpretability focuses on understanding how specific mechanisms within the model contribute to its predictions. Techniques like causal tracing (Meng et al., 2023) provide insights into how neural networks, especially LLMs like GPT, represent and utilize factual knowledge. By locating factual associations, researchers can identify which parts of the model are responsible for generating specific factual outputs.

While local analysis provides insights into specific model decisions, global analysis offers a more holistic view of the model's overall behavior. These two approaches are complementary: local analysis helps interpret individual predictions, while global analysis aids in understanding how the model generalizes across different tasks and domains.

2.2 Neural ODEs in LLMs

Neural Ordinary Differential Equations (ODEs) have emerged as a powerful framework for continuous-time modeling, with applications across various domains, including language models (LLMs).

Introduced by Chen et al. (2019) (Chen et al., 2019), Neural ODEs model the evolution of latent variables as a continuous-time dynamical system. Unlike traditional neural networks, which rely on discrete layers, Neural ODEs define a differential equation parameterized by neural networks, allowing for flexible representations of data evolving over time. Neural ODEs are particularly useful for modeling the temporal patterns seen in language processing, enabling the model to adaptively learn how different linguistic structures evolve (Rubanova et al., 2019).

Additionally, integrating Neural ODEs with attention mechanisms has led to scalable LLMs capable of processing and interpreting real-time sensor data (Wang et al., 2024). A promising direction is combining Neural ODEs with Transformer architectures, which has revealed natural correspondences between Neural ODEs and Transformer attention mechanisms, offering new insights into deep learning models (Hashimoto et al., 2024).

However, despite their potential, current applications of Neural ODEs in understanding neural models often fall short in directly correlating these dynamics with specific input-output relationships, particularly in complex architectures like LLMs. Existing frameworks typically do not address how these learned dynamics can be tuned based on external objectives, such as ensuring fairness or robustness. This presents a significant challenge in applying Neural ODEs to real-world LLM applications.

2.3 Control Theory in LLMs

Control theory offers critical insights into the dynamics and optimization of complex systems, making it highly relevant for improving the interpretability and reliability of Large Language Models (LLMs). The application of robust control helps address uncertainty within these models and ensures they meet performance standards.

Control is crucial in LLM research. As noted by (Liang et al., 2024), controllable text generation aims to generate text according to specific requirements, including content and attribute control. Fine-tuning and retraining adjust the model during training, while reinforcement learning and prompt engineering guide the model during inference to enhance text controllability. Researchers in control engineering (Kevian et al., 2024) use specialized datasets and evaluation methods to understand the problem-solving capabilities of different advanced LLMs in control engineering contexts, guiding future improvements.

Control theory in LLMs is an evolving field. By exploring text generation, safety, multimodal tasks, and domain-specific applications, control theory can make LLMs more interpretable, reliable, and powerful, with broad applications in areas such as autonomous systems, healthcare, and finance.

3 Contributions and Structure of the Paper

This paper introduces an innovative approach for enhancing the interpretability and reliability of Large Language Models (LLMs) by integrating Neural Ordinary Differential Equations (Neural ODEs) with robust control mechanisms. The key contributions are as follows:

- **Innovative Integration Method**: We pioneer the integration of Neural ODEs and robust control for LLMs, offering a new perspective on analyzing input-output relationships in LLMs that has not been explored before.
- **Enhanced Model Understanding**: This work enables the transformation of LLM inputs and outputs into a lower-dimensional latent space, providing a means to study internal information-processing pathways, which significantly enhances model interpretability.
- **Improved Output Quality**: The application of robust control ensures that LLM outputs meet specific performance criteria, improving their quality and reliability for practical use.

The paper is structured as follows: Section 4 utilizes Ordinary Differential Equations (ODEs) to model LLM processes, offering a continuous and interpretable framework, with robust control mechanisms introduced to enhance output reliability and ensure ethical standards. Section 5 presents the methodological framework, integrating Neural ODEs with and without control mechanisms to model dynamic processes within LLMs. Section 6 provides a comparative analysis of Neural ODE models, with and without control, focusing on training/validation loss, prediction accuracy, and latent space dynamics. Finally, Section 7 demonstrates how integrating control mechanisms into Neural ODEs enhances LLM stability and generalization, which is crucial for developing trustworthy AI in high-stakes domains, setting the stage for future research on transparent and accountable AI technologies.

4 Theoretical Framework

4.1 Neural ODEs

Because Ordinary Differential Equations (ODEs) are inherently unable to model text directly, we need to map the input and output of Large Language Models (LLMs) into a latent space. This latent space representation allows us to work with continuous-time dynamics, enabling the use of Neural ODEs for modeling the evolution of LLM inputs and outputs in a more flexible and accurate manner. Neural Ordinary Differential Equations (Neural ODEs) offer a powerful framework for modeling continuous-time dynamics. Their application in large language models (LLMs) provides a way to better capture the temporal relationships inherent in language data. Unlike traditional discrete models (e.g., RNNs or LSTMs), which treat sequences of data as a series of steps, Neural ODEs model the evolution of latent variables continuously over time, offering a more flexible and accurate representation of sequential data.

In the context of LLMs, we define the hidden state $h(t)$ as a function of time t. The evolution of this state is described by the following equation:

$$\frac{dh(t)}{dt} = f(t, h(t), \theta) \tag{1}$$

where f is a function parameterized by a neural network. For simplicity, assume that f is a single-layer neural network with a linear transformation followed by an activation function σ:

$$f(t, h(t), \theta) = \sigma(W \cdot h(t) + b) \tag{2}$$

Here, W is the weight matrix, b is the bias vector, and θ represents the parameters of the network.

This continuous modeling approach is particularly useful for time-series and language processing tasks, where data evolves over time. To solve the differential equation, numerical methods like Euler's method can be applied:

$$h(t + \Delta t) \approx h(t) + \Delta t \cdot f(t, h(t), \theta) \tag{3}$$

This update rule models how the hidden state evolves over small time steps, enabling the model to learn how language structures change dynamically. Compared to traditional discrete models like RNNs or LSTMs, Neural ODEs offer several advantages, including more natural modeling of continuous temporal patterns and flexibility in capturing complex dependencies over time.

The mapping of LLM inputs and outputs to a latent space allows us to model the input-output relationships using continuous-time dynamics. Specifically, the input sequence $\mathbf{X} = [x_1, x_2, \ldots, x_T]$ is

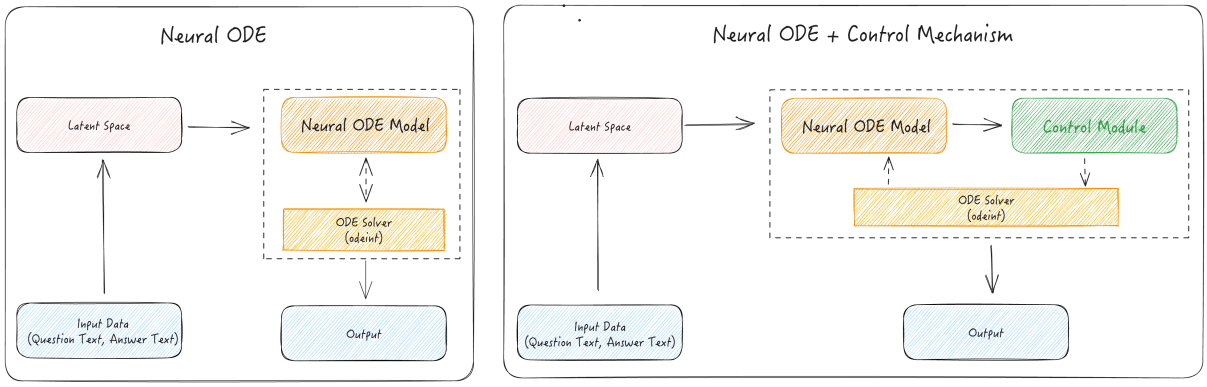

Figure 1: Model Architecture for Methodology

first embedded into a latent space $\mathcal{Z}$ via an embedding function ϕ:

$$\mathcal{Z}_t = \phi(\mathbf{X}_t) \tag{4}$$

where $\mathcal{Z}_t$ represents the embedded representation of the input at time step t, and ϕ is the function mapping the raw input tokens into the latent space. The output sequence $\mathbf{Y} = [y_1, y_2, \ldots, y_T]$ is then modeled in the same latent space, allowing the Neural ODE to capture the temporal evolution and dynamic mapping between the inputs and the corresponding outputs.

This formulation enables Neural ODEs to model the continuous-time evolution of LLM states, offering a powerful tool for handling sequential data with complex dependencies, such as language. By operating in the latent space, the Neural ODE framework is able to handle the high-dimensional, variable-length sequences typical of language models more efficiently and flexibly.

4.2 Control Mechanism

To improve the reliability of the LLM outputs, we introduce a robust control mechanism. The goal is to minimize the difference between the model's output y and the desired output $y_{desired}$. The cost function J is defined as:

$$J = \sum_{i=1}^{n} w_i \cdot L_i(y, y_{desired}) \tag{5}$$

where L_i represents the individual loss components, and w_i are their respective weights. This cost function quantifies the discrepancy between the predicted and desired outputs across different loss components.

The output y depends on the hidden state h and control input u, and is given by:

$$y = g(h, u) \tag{6}$$

where $g(h, u) = V \cdot (h + u) + c$ is a neural network-based function, with V and c as additional parameters. The control input u is adjusted during training to minimize the cost function J. This can be formulated as:

$$\begin{aligned} u^*(t) &= \arg\min_u J(u) \\ &= \arg\min_u \sum_{i=1}^{n} w_i \cdot L_i(g(h, u), y_{desired}) \end{aligned} \tag{7}$$

By optimizing the control input u, we ensure that the model's output y closely aligns with the desired output. This control mechanism enhances the LLM's stability and reliability, making it more robust in real-world applications.

4.3 Integrating Neural ODE and Control Mechanism

To further enhance the LLM's performance, we integrate the Neural ODE with the control mechanism. By incorporating the control input u into the Neural ODE, the evolution of the hidden state $h(t)$ is modeled as:

$$\frac{dh(t)}{dt} = f(t, h(t), \theta, u) \tag{8}$$

In this combined framework, the cost function J depends on the entire sequence of outputs $\{y(t)\}$, where $y(t) = g(h(t), u(t))$ for $t = 0, \Delta t, \ldots, T$. The optimization problem is then formulated as:

$$(u^*, h^*) = \arg\min_{u,h} J(\{g(h(t), u(t))\}_{t=0}^{T}) \tag{9}$$

This integrated approach ensures that the LLM's output reflects both the internal processing of input data and the external control criteria. By optimizing both the hidden state and the control input, we can improve the model's stability, generalization, and adaptability to various tasks.

The combination of Neural ODEs and control mechanisms allows for a more flexible and interpretable model. This approach is particularly beneficial for high-stakes applications where the model needs to be both interpretable and reliable. Furthermore, the continuous-time nature of Neural ODEs, combined with the control theory, provides a robust framework for optimizing model performance across a wide range of tasks, including real-time language processing and decision-making in dynamic environments.

4.4 Conclusion

In this section, we introduced a theoretical framework that integrates Neural ODEs with control mechanisms to optimize the performance and interpretability of LLMs. This approach provides flexibility in modeling sequential data and ensures that the model output meets specific performance criteria. Additionally, exploring how this framework can be adapted to handle diverse types of language data, such as multimodal inputs or noisy real-world data, would significantly expand its applicability. Further investigations into the interpretability of the combined model could also help provide deeper insights into how both internal dynamics and control inputs contribute to the model's decision-making process.

5 Methodology

Building on the theoretical foundations established in the previous section, this section presents two algorithmic frameworks designed to enhance the interpretability and reliability of large language models (LLMs). These frameworks integrate Neural ODEs (Ordinary Differential Equations) and robust control mechanisms. The first framework utilizes Neural ODEs alone, encoding LLM inputs and outputs into a latent space and evolving the state using advanced optimization techniques. The second framework incorporates control mechanisms into the Neural ODEs, allowing for dynamic adjustment of the model's state to achieve stable and reliable outputs. Both frameworks aim to improve the transparency and performance of LLMs by revealing their continuous and dynamic transformations.

5.1 Neural ODE for LLM Input-Output Mapping

The first algorithm uses a basic Neural ODE framework to model the input-output relationships in LLMs without any control mechanisms. This framework consists of three key components, detailed in Algorithm 1.

Algorithm 1 Train Neural ODE for LLM Input-Output Mapping

Input: Dataset (Q, A), Parameters θ, Learning rate α, Epochs E
Output: Optimized parameters θ^*
Initialize model $\mathcal{M} \leftarrow \text{NeuralODE}(\theta)$
Initialize optimizer $Opt \leftarrow \text{Adam}(\mathcal{M}, \alpha)$
for epoch $= 1$ to E **do**
  for each $(q, a) \in (Q, A)$ **do**
    $h \leftarrow \text{Integrate}(\mathcal{M}, q)$
    $loss \leftarrow \text{MSE}(h, a)$
    $Opt.\text{step}(loss)$
  end for
end for
Return: θ^*

The algorithm models the relationships between inputs and outputs using Neural ODEs without incorporating any additional control mechanisms. The architecture consists of three key components: the input layer, which transforms raw input tokens into a lower-dimensional latent space using an embedding layer; the Neural ODE block, which models the hidden state dynamics as a continuous-time evolution governed by the Neural ODE; and the output layer, which maps the final hidden state to the desired output using a fully connected (dense) layer followed by an activation function.

5.2 Neural ODE with Control Mechanism

The second algorithm incorporates a control mechanism into the Neural ODE framework. This framework is designed to dynamically adjust the model's hidden state and ensure output reliability. The key components of this model are outlined in Algorithm 2.

The algorithm integrates a robust control mechanism into the Neural ODE framework, dynamically adjusting the hidden state to improve output reliability. The architecture includes the input layer, which transforms raw input tokens into embeddings; the Neural ODE block with control, which

Algorithm 2 Train Neural ODE with Control for LLMs

Input: Dataset (Q, A), Parameters θ, Learning rate α, Epochs E, Control type c
Output: Optimized parameters θ^*
Initialize model $\mathcal{M} \leftarrow \text{NeuralODE}(\theta)$
Initialize optimizer $Opt \leftarrow \text{Adam}(\mathcal{M}, \alpha)$
for epoch $= 1$ to E **do**
  for each $(q, a) \in (Q \cup Q_{test}, A \cup A_{test})$ **do**
    $h \leftarrow q$ {Initialize hidden state}
    for t $= 1$ to T **do**
      $f \leftarrow \text{Dynamics}(h, \theta)$ {Compute dynamics}
      $H \leftarrow \text{Control}(h, q, c)$ {Apply control}
      $h \leftarrow h + \Delta t \cdot (f + H)$ {Update hidden state}
    end for
    $loss \leftarrow \text{MSE}(h, a)$ {Compute loss}
    $Opt.\text{step}(loss)$ {Optimize model parameters}
  end for
end for
Return: θ^*

models hidden state dynamics while incorporating a control input $u(t)$ to guide the state evolution; the control module, which computes the optimal control input based on the current hidden state and predefined standards; and the output layer, which maps the controlled hidden state to the desired output.

6 Experiment and Results

The experiments aim to validate these two methods' contributions to improving model performance in various contexts. Specifically, we conduct two experiments:

Experiment I visualizes the input-output relationships using Neural ODEs across multiple question-answer (QA) datasets from diverse domains, demonstrating the ability of Neural ODEs to capture complex input-output dynamics. Experiment II, on the other hand, applies Control Theory to regulate LLM outputs, assessing the impact of control mechanisms on model performance and reliability, particularly in terms of consistency and stability.

6.1 Experimental Data

Experiment I For Experiment I, we selected six distinct QA datasets that cover a range of domains, including factual knowledge bases, commonsense reasoning tasks, medical information, mathematical problem-solving, and truthful response generation. These datasets allow us to assess the versatility and adaptability of Neural ODEs in diverse settings. Table 2 provides an overview of these datasets, detailing their repositories, sizes, and specific tasks.

Experiment II For Experiment II, we utilized the `aligner/aligner-20K` dataset, which consists of 20,000 aligned QA pairs carefully curated to ensure high relevance and accuracy between the input questions and their corresponding answers. This dataset is ideal for assessing the role of Control Theory in stabilizing model outputs. Experiment II is designed to evaluate the effectiveness of Control Theory in regulating model outputs and improving consistency.

6.2 Experiment I: Results and Analysis

For Experiment I, the Neural ODE model was trained on six different QA datasets. The training losses across epochs were recorded to assess the model's convergence behavior. Table 1 presents the training and validation losses for each dataset.

Table 1: Training and Validation Losses at Epoch 30 for QA Datasets in Experiment I

Dataset Name	Training Loss	Validation Loss
commonsense_qa	0.0290	0.0610
GammaCorpus-fact-qa	0.0278	0.0578
medical-qa	0.0053	0.0224
rvv-karma_Math-QA	0.0048	0.0086
trivia_qa	0.0291	0.0597
TruthfulQA	0.0052	0.0339

Principal Component Analysis (PCA) was used to reduce the dimensionality of the embedding vectors to two dimensions for visualization purposes. Figure 2 shows the PCA projections of embeddings from all six datasets, highlighting the clustering patterns and input-output transformation across diverse datasets.

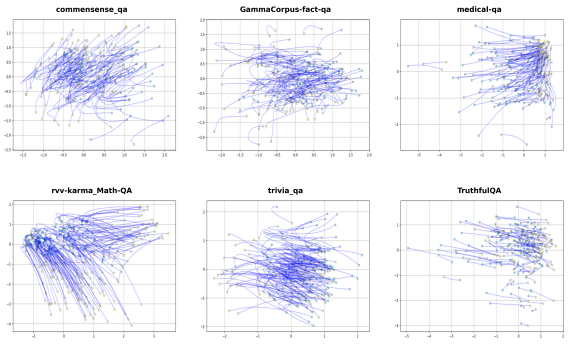

Figure 2: Input-to-Output Transformation Diagram Across Various QA Datasets (Yellow represents the starting point, and green represents the ending point)

Dataset Feature Analysis The trajectory distributions for `commonsense_qa` and `trivia_qa` are relatively dispersed, indicating that the input-output mappings in commonsense reasoning tasks are complex, with diverse transformation paths. On the other hand, `medical-qa` shows a more concentrated endpoint distribution (with yellow points clustered on the right), suggesting that answers in the medical domain may have a more standardized format or structure. The `rvv-karma_Math-QA` dataset exhibits a distinctly radial distribution, which likely reflects the logical reasoning path involved in solving mathematical problems. Lastly, the trajectories for `TruthfulQA` are relatively short and evenly distributed, suggesting that the transformation process for truthfulness verification is more straightforward.

Common Features All datasets demonstrate a continuous transformation process from input (yellow points) to output (blue points), validating that Neural ODEs can effectively model the dynamic characteristics of LLMs. The trajectories generally exhibit nonlinear features, indicating that the input-output transformation in these QA tasks is complex.

Conclusion In terms of modeling effectiveness, Neural ODE successfully captured the dynamic features of different QA tasks across various domains. The model was able to adapt to the specific characteristics of different datasets, demonstrating its versatility.

Regarding domain differences, the input-output transformations for different QA tasks revealed unique patterns. Specialized domains, such as medical and mathematical QA, showed more structured and regular transformation paths compared to commonsense reasoning tasks. This suggests that specialized fields benefit from more predictable and structured transformations, while more general domains, such as commonsense reasoning, involve more complex mappings.

6.3 Experiment II: Results and analysis

Experiment II examines the impact of Control Theory on the model's training process. We applied four control strategies, as detailed in the appendix, and compared the results with and without the use of control mechanisms. PCA was employed to visualize the output embeddings generated by the model in both scenarios, with and without the application of Control Theory.

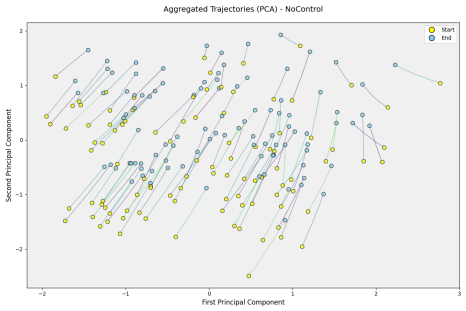

Figure 3: Trajectory Plots without Control

Trajectory Feature Analysis The analysis of the five trajectory plots—NoControl, LQRControl, MPCControl, RLControl, and SMControl—reveals several key patterns in the control mechanisms' performance. In the NoControl scenario, the trajectories appear scattered with no clear pattern, and the path from start to end is irregular. The absence of clear directionality and the significant variation in trajectory length highlight the instability of the model without control mechanisms.

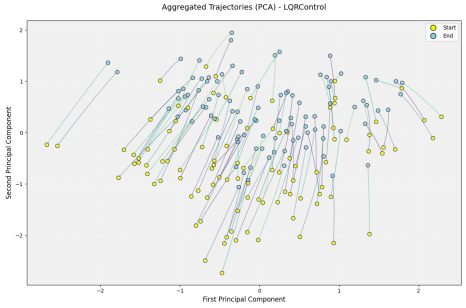

Figure 4: Trajectory Plots with LQR Control

When LQRControl is applied, the trajectories become notably smoother. The endpoint distribu-

tion is more concentrated, reflecting the symmetrical characteristics typical of linear control. This demonstrates that LQRControl provides more consistent results than NoControl.

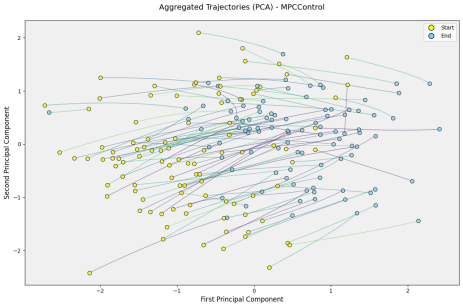

Figure 5: Trajectory Plots with MPC Control

With MPCControl, the trajectories exhibit a predictive feature, where the path planning becomes more rational, and the trajectory is more coherent. The endpoint distribution is moderate, indicating that MPCControl achieves a balanced optimization. The model demonstrates characteristics of model predictive control, where future trajectories are accounted for, guiding the model toward better output predictions.

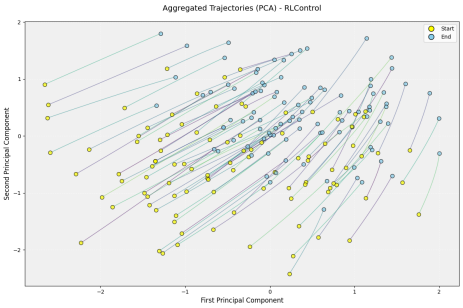

Figure 6: Trajectory Plots with RL Control

RLControl, in contrast, shows a stronger directionality. The model's adaptation through reinforcement learning allows it to learn an effective control strategy, leading to a more concentrated endpoint distribution. The trajectories in this case reflect the self-adaptive nature of reinforcement learning, as the model learns and refines its control policy over time.

SMControl stands out as the most orderly and regular, with a clear sliding mode surface feature evident in the trajectories. The control effect is the most stable, and the directionality is the clearest, reflecting the strengths of sliding mode control in ensuring both stability and clarity in the model's output.

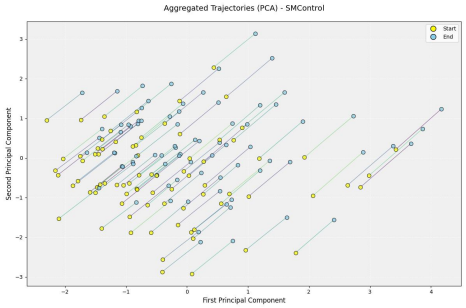

Figure 7: Trajectory Plots with SM Control

Conclusions In terms of overall evaluation, the application of control theory significantly enhances the model's output quality. Each control strategy exhibits its own unique advantages: SMControl excels in stability, RLControl in adaptability, MPCControl in prediction, and LQRControl in linear problems. These findings confirm that control theory plays a crucial role in enhancing both the reliability and interpretability of LLM outputs.

7 Conclusion and Future Research

This paper introduces a novel framework that integrates Neural Ordinary Differential Equations (Neural ODEs) with robust control theory to enhance the interpretability and reliability of Large Language Models (LLMs). Our empirical and theoretical analysis demonstrates that combining these approaches significantly improves model performance, stability, and adaptability. By implementing various control strategies, such as LQR, MPC, SM, and RL-based control, we validate their respective strengths in regulating LLM outputs, particularly in dynamic and complex environments.

Looking forward, future research can explore the development of hybrid control mechanisms, scalability optimizations for large-scale deployment, and the integration of advanced interpretability techniques. Additionally, addressing the computational challenges and enhancing generalization across diverse tasks will be crucial in refining this framework. By extending the applications to multimodal systems and real-time language processing, we can ensure that LLMs remain both reliable and interpretable, paving the way for safer and more effective AI systems in real-world applications.

8 Limitation

The proposed framework integrating Neural ODEs with control theory offers significant advancements

in enhancing the interpretability and reliability of Large Language Models (LLMs). However, several limitations remain. First, the computational complexity of integrating continuous-time modeling with control mechanisms introduces substantial overhead, necessitating further optimization for large-scale models. Second, the task of parameter tuning for control strategies, such as feedback gains or prediction horizons, is challenging and requires further research to identify optimal settings across diverse tasks. Additionally, while the framework has shown promising results on specific datasets, its generalization across different LLM architectures and domains remains to be fully validated. Finally, although control theory improves model stability, further attention must be paid to ethical concerns such as fairness and accountability to ensure the responsible deployment of LLMs in high-stakes applications.

9 Acknowledgements

During the writing of this article, generative artificial intelligence tools were used to assist in language polishing and literature retrieval. The AI tool helped optimize the grammatical structure and expression fluency of limited paragraphs, and assisted in screening research literature in related fields. All AI-polished text content has been strictly reviewed by the author to ensure that it complies with academic standards and is accompanied by accurate citations. The core research ideas, method design and conclusion derivation of this article were independently completed by the author, and the AI tool did not participate in the proposal of any innovative research ideas or the creation of substantive content. The author is fully responsible for the academic rigor, data authenticity and citation integrity of the full text, and hereby declares that the generative AI tool is not a co-author of this study.

References

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

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2019. Neural ordinary differential equations. *Preprint*, arXiv:1806.07366.

Koji Hashimoto, Yuji Hirono, and Akiyoshi Sannai. 2024. Unification of symmetries inside neural networks: Transformer, feedforward and neural ode. *Preprint*, arXiv:2402.02362.

Darioush Kevian, Usman Syed, Xingang Guo, Aaron Havens, Geir Dullerud, Peter Seiler, Lianhui Qin, and Bin Hu. 2024. Capabilities of large language models in control engineering: A benchmark study on gpt-4, claude 3 opus, and gemini 1.0 ultra. *Preprint*, arXiv:2404.03647.

Xun Liang, Hanyu Wang, Yezhaohui Wang, Shichao Song, Jiawei Yang, Simin Niu, Jie Hu, Dan Liu, Shunyu Yao, Feiyu Xiong, and Zhiyu Li. 2024. Controllable text generation for large language models: A survey. *Preprint*, arXiv:2408.12599.

Zachary C. Lipton. 2017. The mythos of model interpretability. *Preprint*, arXiv:1606.03490.

Scott Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Preprint*, arXiv:1705.07874.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2023. Locating and editing factual associations in gpt. *Preprint*, arXiv:2202.05262.

Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. 2019. Latent odes for irregularly-sampled time series. *Preprint*, arXiv:1907.03907.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. *Preprint*, arXiv:1703.01365.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. 2019. What do you learn from context? probing for sentence structure in contextualized word representations. *Preprint*, arXiv:1905.06316.

Jesse Vig. 2019. Visualizing attention in transformer-based language representation models. *Preprint*, arXiv:1904.02679.

Cong Wang, Aoming Liang, Fei Han, Xinyu Zeng, Zhibin Li, Dixia Fan, and Jens Kober. 2024. Learning adaptive hydrodynamic models using neural odes in complex conditions. *Preprint*, arXiv:2410.00490.

Table 2: Overview of QA Datasets Used in Experiment I

Dataset Name	Domain
commonsense_qa	Commonsense Reasoning
GammaCorpus-fact-qa	General Knowledge, Fact-Checking
medical-qa	Medical, Healthcare
rvv-karma_Math-QA	Mathematics, Logical Reasoning
trivia_qa	Trivia, Common Knowledge
TruthfulQA	Truthfulness, Ethics

A Detailed Experiment

B Appendix:Control Strategies Summary

In this section, we present four control strategies implemented for regulating the outputs of Large Language Models (LLMs): Linear Quadratic Regulator (LQR), Model Predictive Control (MPC), Sliding Mode Control (SMC), and Reinforcement Learning (RL) based control. Each of these methods employs different approaches to improve the model's performance, stability, and adaptability.

B.1 Linear Quadratic Regulator Control (LQRControl)

Linear Quadratic Regulator (LQR) is an optimal control strategy that minimizes a quadratic cost function to stabilize the system. The objective is to penalize both the state deviations and the control effort. The control input u is calculated as the negative feedback of the error between the desired state q and the model output y, scaled by the feedback gain matrix K.

The LQRControl class has three primary parameters: - $\mathbf{Q} \in \mathbb{R}^{n \times n}$: A state penalty matrix that penalizes deviations of the system state from the desired state. - $\mathbf{R} \in \mathbb{R}^{m \times m}$: A control effort penalty matrix that penalizes large control actions. - $\mathbf{K} \in \mathbb{R}^{n \times m}$: A feedback gain matrix that defines how much influence the control input has over the system's state.

The control law is defined as follows:

$$u = -\mathbf{K} \cdot (q - y) \cdot \sigma \left(\sum \mathbf{Q} \right) / \left(\sum \mathbf{R} + \epsilon \right)$$

where σ is the sigmoid activation function, and ϵ is a small constant to avoid division by zero.

LQRControl is effective in stabilizing systems where the relationship between the state and control is linear, and it is particularly useful in scenarios that require precise error correction.

B.2 Model Predictive Control (MPCControl)

Model Predictive Control (MPC) uses a model of the system to predict future states and optimize the control input over a finite time horizon. MPC computes the control input by solving an optimization problem that minimizes a cost function over the predicted trajectory. This method is highly effective in systems where future behavior can be predicted and adjusted.

The MPCControl class uses the following parameters: - $\mathbf{Horizon} \in \mathbb{N}$: The prediction horizon over which the system's behavior is forecasted. - $\mathbf{StatePredictor}$: A neural network used to predict future system states based on the current state and past control inputs. - $\mathbf{Controller}$: A neural network that computes the optimal control action by considering the predicted states.

The MPC control input is computed as:

$$u = \arg\min_{u} \sum_{t=1}^{\text{Horizon}} \left(\|y_t - q_t\|^2 + \lambda_u \|u_t\|^2 \right)$$

where y_t is the predicted state at time step t, q_t is the desired state, and λ_u is a regularization parameter.

MPCControl is particularly suitable for systems that require optimization over a planning horizon, such as robotics and autonomous systems.

B.3 Sliding Mode Control (SMControl)

Sliding Mode Control (SMC) is a robust control technique designed to handle systems with uncertainties or disturbances. SMC forces the system state to "slide" along a predefined sliding surface, ensuring robustness and stability. The control input is determined based on the system's error and the sliding surface function, which enforces the desired behavior.

The SMControl class involves the following parameters: - $\lambda \in \mathbb{R}^n$: The scaling factor for the sliding surface, which governs the control input's sensitivity. - $\eta \in \mathbb{R}^n$: A parameter that adjusts the system's response to the error. - $\phi \in \mathbb{R}^n$: A parameter that controls the non-linearity in the sliding surface.

The control law is defined as:

$$u = \lambda \cdot s + k \cdot \text{sat}(s/\phi)$$

where $s = q - y$ is the error, and $\text{sat}(\cdot)$ is the saturation function, typically $\text{sat}(x) = \tanh(x)$.

SMControl is effective for systems requiring high robustness against uncertainties and disturbances, such as in automotive or aerospace applications.

B.4 Reinforcement Learning Control (RLControl)

Reinforcement Learning (RL) Control leverages reinforcement learning to adaptively learn the optimal control policy based on feedback from the environment. The RL controller utilizes a value network to estimate the expected future reward and a policy network to decide the control actions. The advantage function adjusts the control input by calculating the difference between expected and actual outcomes.

The RLControl class includes the following components: - $\gamma \in [0, 1]$: The discount factor that determines the importance of future rewards. - $\mathbf{ValueNetwork}$: A neural network that estimates the value of the current state. - $\mathbf{PolicyNetwork}$: A neural network that generates the control action based on the current state and the desired state.

The control input is computed as:

$$u = \text{policy}(y, q) \cdot \sigma(\|q - y\| - \text{value}(y))$$

where σ is the sigmoid activation function, $\|q - y\|$ is the norm of the error, and $\text{value}(y)$ is the output of the value network.

RLControl is suitable for complex, dynamic environments where the model must continuously learn and adapt to new situations.

B.5 Conclusion

Each control strategy offers distinct advantages depending on the application scenario. LQRControl is well-suited for linear systems with simple control tasks, while MPCControl provides superior performance in systems requiring trajectory optimization and prediction. SMControl excels in environments that demand robustness and stability, and RLControl is ideal for tasks that require adaptive learning and optimization in complex, dynamic settings. The integration of these control strategies demonstrates substantial improvements in model performance, stability, and interpretability across a variety of tasks.