Mechanistic Unveiling of Transformer Circuits: Self-Influence as a Key to Model Reasoning

Lin Zhang[1,2,3*], **Lijie Hu**[1,2], **Di Wang**[1,2†]
[1]Provable Responsible AI and Data Analytics (PRADA) Lab
[2]King Abdullah University of Science and Technology
[3]Harbin Institute of Technology, Shenzhen
23s058005@stu.hit.edu.cn
{lijie.hu, di.wang}@kaust.edu.sa

Abstract

Transformer-based language models have achieved significant success; however, their internal mechanisms remain largely opaque due to the complexity of non-linear interactions and high-dimensional operations. While previous studies have demonstrated that these models implicitly embed reasoning trees, humans typically employ various distinct logical reasoning mechanisms to complete the same task. It is still unclear which multi-step reasoning mechanisms are used by language models to solve such tasks. In this paper, we aim to address this question by investigating the mechanistic interpretability of language models, particularly in the context of multi-step reasoning tasks. Specifically, we employ circuit analysis and self-influence functions to evaluate the changing importance of each token throughout the reasoning process, allowing us to map the reasoning paths adopted by the model. We apply this methodology to the GPT-2 model on a prediction task (IOI) and demonstrate that the underlying circuits reveal a human-interpretable reasoning process used by the model.

1 Introduction

In recent years, the Transformer architecture, introduced by (Vaswani et al., 2017), has become an efficient neural network structure for sequence modeling (Brown et al., 2020). Previous research (Hou et al., 2023; Dong et al., 2021) has confirmed that large models rely primarily on reasoning rather than mere memorization when answering questions. However, the "thought process" of these models remains unclear, as shown in Figure 1. How can we explore the thought process employed by large models during reasoning (Wei et al., 2022; Kojima et al., 2022)? Addressing this question is not only crucial for deepening our understanding of these models but also essential for developing the next generation of reliable language-based reasoning systems (Creswell and Shanahan, 2022; Creswell et al., 2022; Chen et al., 2023; Hu et al., 2024a; Cheng et al., 2024; Yang et al., 2024a).

Influence functions focus on analyzing models from the perspective of their training data (Koh and Liang, 2017). They have been shown to be versatile tools applicable to a wide range of tasks, including understanding model behavior, debugging models, detecting dataset errors, and generating visually indistinguishable adversarial examples(Hu et al., 2024b,c). As a variant of influence functions, self-influence is a technique for evaluating the impact of specific inputs within a neural network on the model's output. For different tokens within an input sample, self-influence scores reflect the significance of each token across various layers of the model. By tracking the influence changes of different tokens throughout the reasoning process of large language models (LLMs), it is possible to map the thought process executed by the model. However, directly calculating self-influence for all parameters in LLMs is practically infeasible due to the enormous computational resources and substantial memory consumption required.

As a result, circuit analysis within the framework of Mechanistic Interpretability (MI) (Olah, 2022; Nanda, 2023) has become a focal point of research. MI aims to discover, understand, and verify the algorithms encoded in model weights by reverse engineering the model's computations into human-understandable components (Meng et al., 2022; Geiger et al., 2021; Geva et al., 2020; Zhang et al., 2024; Hong et al., 2024). A key method in this field is circuit analysis (Conmy et al., 2023; Olah et al., 2020). In this approach, neural networks are conceptualized as computational graphs, where circuits represent sub-graphs composed of interconnected features and the weights that link them. These circuits function as fundamental computational units and building blocks of the network

[*]Work done during an internship at PRADA Lab.
[†]Corresponding author.

Findings of the Association for Computational Linguistics:
NAACL 2025, pages 1387–1404

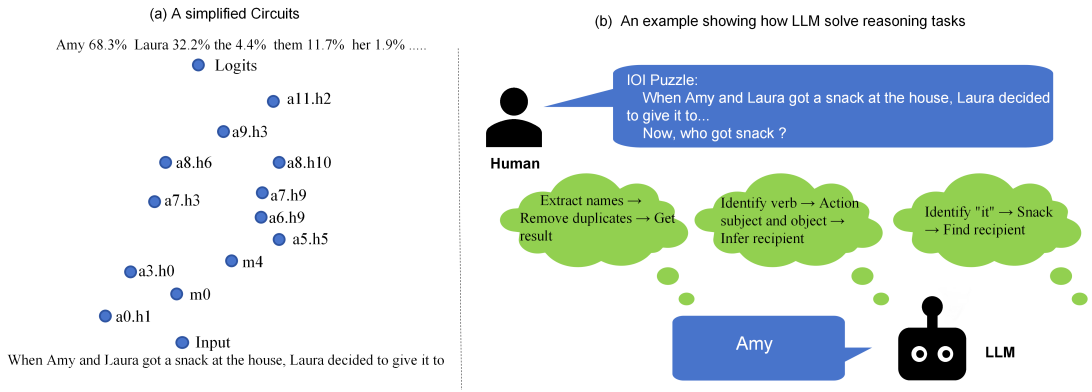

Figure 1: (a) A simplified illustration of circuits within the model. (b) An example of how a language model (LLM) tackles a reasoning task, such as the Indirect Object Identification (IOI) puzzle. The model identifies key entities, actions, and pronouns to deduce the recipient. The reasoning process involves steps like extracting names, determining actions, and linking pronouns to objects to reach the answer "Amy."

(Michaud et al., 2024; Olah et al., 2020). Representing the most critical components for completing specific tasks, these circuits capture essential computational processes without the complexity of analyzing the full model.

To address the aforementioned limitations, we propose the Self-Influence Circuit Analysis Framework (SICAF) as a mechanistic interpretation method to trace and analyze the thought process that language models (LMs) employ during complex reasoning tasks. To facilitate detailed analysis, our investigation proceeds in three stages: (1) identifying the circuit within the model through existing automatic circuit-finding methods, (2) calculating the self-influence of different tokens across various layers of the circuit in each sample, and (3) deducing the thought process that the language model employs during reasoning by analyzing changes in self-influence scores across layers.

As a preliminary step, we employ the automatic circuit-finding methods EAP, EAP-IG, and EAP-IG-KL (Hanna et al., 2024) with varied parameters to identify circuits within a (finetuned) GPT-2 model (Radford et al., 2019) that performs a specific natural language task, indirect object identification (IOI) (Wang et al., 2022). We find that these circuits are small (containing 1-2% of edges) and faithful (recovering $\geq$85% of model performance). Under the same parameter constraints, EAP-IG identifies circuits that are more faithful. Furthermore, the nodes within these circuits are primarily concentrated in the first layer and the last few layers of the model.

Next, we conduct a detailed examination of the nodes within the identified circuits, calculating the self-influence of different tokens across various layers of each sample and inferring the reasoning process the language model employs by analyzing how self-influence scores change across layers.

By focusing on a specific task within the (fine-tuned) GPT-2 model, we have gained several key insights into the challenges of mechanistic interpretability in transformer-based language models. In particular:

- We propose a new mechanistic interpretation framework, SICAF, to trace and analyze the thought process that language models (LMs) employ during complex reasoning tasks.
- By employing various methods to identify and analyze circuits within the model, we observed a consistent pattern: the model's key parameters are primarily concentrated in the first layer and the final few layers.
- We extend SICAF by applying multiple Circuit Analysis methods to uncover and analyze diverse thought processes embedded in different circuits of the model.

2 Related Work

Interpretability Methods in Language Models. Interpretability paradigms for AI decision-making range from black-box techniques, which focus on input-output relationships, to internal analyses that delve into model mechanics (Bereska and Gavves, 2024). Behavioral interpretability (Warstadt et al., 2020; Covert et al., 2021; Casalicchio et al., 2019) treats models as black boxes, examining robustness and variable dependencies, while attributional interpretability (Sundararajan et al., 2017; Smilkov et al., 2017; Shrikumar et al., 2017) traces outputs

back to individual input contributions. Concept-based interpretability (Belinkov, 2022; Burns et al., 2023; Zou et al., 2023; Yang et al., 2024b; Hu et al.) explores high-level concepts within models' learned representations. In contrast, mechanistic interpretability (Bereska and Gavves, 2024) adopts a bottom-up approach, analyzing neurons, layers, and circuits to uncover causal relationships and precise computations, offering a detailed understanding of the model's internal operations.

Circuit Analysis. Neural networks can be conceptualized as computational graphs, where circuits of linked features and weights serve as fundamental computational units (Bereska and Gavves, 2024). Recent research has focused on dissecting models into interpretable circuits. Automated Circuit Discovery (ACDC) (Conmy et al., 2023) automates a large portion of the mechanistic interpretability workflow, but it is inefficient due to its recursive nature. Syed et al. (2023) introduced Edge Attribution Patching (EAP) to identify circuits for specific tasks, while Hanna et al. (2024) introduced EAP with integrated gradients(EAP-IG), which improves upon EAP by identifying more faithful circuits. Circuit analysis leverages key task-relevant parameters (Bereska and Gavves, 2024) and feature connections (He et al., 2024) within the network to capture core computational processes and attribute outputs to specific components (Miller et al., 2024), bypassing the need to analyze the entire model. This approach maintains efficiency and scalability, offering a practical alternative for understanding model behavior.

Influence Function. The influence function, initially a staple in robust statistics (Cook, 2000; Cook and Weisberg, 1980), has seen extensive adoption within machine learning since Koh and Liang (2017) introduced it to the field. Its versatility spans various applications, including detecting mislabeled data, interpreting models, addressing model bias, and facilitating machine unlearning tasks. Notable works in machine unlearning encompass unlearning features and labels (Warnecke et al., 2023), minimax unlearning (Liu et al., 2024), forgetting a subset of image data for training deep neural networks (Golatkar et al., 2020, 2021), graph unlearning involving nodes, edges, and features. Recent advancements, such as the LiSSA method (Agarwal et al., 2017; Kwon et al., 2023) and kNN-based techniques (Guo et al., 2021), have been proposed to enhance computational efficiency. Besides, various studies have applied influence functions to interpret models across different domains, including natural language processing (Han et al., 2020) and image classification (Basu et al., 2021), while also addressing biases in classification models (Wang et al., 2019), word embeddings (Brunet et al., 2019), and finetuned models (Chen et al., 2020). Despite numerous studies on influence functions, we are the first to apply them to explain the thought process in language models (LMs) during reasoning tasks. We propose a new mechanistic interpretation framework, SICAF, to trace and analyze the reasoning strategies that language models (LMs) employ for complex tasks. Furthermore, compared to traditional neural networks, circuits contain only the most essential parameters of the model, significantly reducing the computational cost of calculating influence functions.

3 Preliminary

Automate Circuit Finding. Edge Attribution Patching (EAP) is a gradient-based method designed to efficiently identify circuits responsible for specific behaviors in neural networks. It estimates the importance of each edge by calculating the change in the model's loss when that edge is corrupted. The score for an edge (u, v) is given by:

$$(z'_u - z_u)^\top \nabla_v L(s) \tag{1}$$

where $\nabla_v L(s)$ is the gradient of the loss function L with respect to the input of node v, and z_u and z'_u represent the clean and corrupted inputs to node u, respectively. EAP-IG extends this approach by incorporating Integrated Gradients, which computes gradients along a linear path between clean and corrupted inputs. The integrated gradients score for an edge (u, v) is:

$$(z'_u - z_u)\frac{1}{m}\sum_{k=1}^{m}\frac{\partial L(z' + \frac{k}{m}(z - z'))}{\partial z_v} \tag{2}$$

where m is the number of steps used to approximate the integral, and z and z' represent the clean and corrupted inputs. EAP-IG addresses the issue of near-zero gradients in important features, providing a more accurate estimation of edge importance and a more faithful representation of the model's behavior. Both methods aim to identify the most crucial edges in a model's circuit, but EAP-IG achieves this with greater precision and reliability. EAP-IG-KL further runs EAP-IG with Kullback-Leibler (KL) divergence as the loss improves upon

EAP-IG by incorporating Kullback-Leibler (KL) divergence to measure the difference between the model's activations on clean and corrupted inputs, ensuring higher fidelity in capturing task-specific behaviors, even under interventions. run EAP-IG with KL divergence as the loss.

Influence Functions. Influence functions provide an efficient approximation for measuring how small perturbations in the training data affect a model's parameters without retraining. For a model with parameters θ^*, minimizing the empirical risk over training data, the influence function evaluates how a slight change in a specific training point z modifies the model's parameters θ_ϵ when its weight is increased by ϵ. The optimization problem is defined as:

$$\theta_\epsilon\{z\} = \arg\min_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(h_\theta(z_i)) + \epsilon\ell(h_\theta(z)) \tag{3}$$

Using a first-order Taylor expansion around θ^*, the new parameters θ_ϵ can be approximated as:

$$\theta_\epsilon \approx \theta^* - \epsilon H_{\theta^*}^{-1}\nabla_\theta \ell(h_{\theta^*}(z)) \tag{4}$$

Where H_{θ^*} is the Hessian matrix of the loss function with respect to the model parameters, and $\nabla_\theta \ell(h_{\theta^*}(z))$ is the gradient of the loss function evaluated at θ^*. The influence function is then given by:

$$I(z) = -H_{\theta^*}^{-1}\nabla_\theta \ell(h_{\theta^*}(z)) \tag{5}$$

Additionally, the influence of a training sample z on a test sample z_t is:

$$I(z, z_t) = -\nabla_\theta \ell(h_{\theta^*}(z_t))^\top H_{\theta^*}^{-1}\nabla_\theta \ell(h_{\theta^*}(z)) \tag{6}$$

This equation measures the approximate change in the test sample's loss when the weight of a training sample is perturbed, offering insight into how training points influence model predictions.

4 Method

We propose a novel mechanistic interpretation framework, SICAF, to trace and analyze the thought process that language models (LMs) employ during complex reasoning tasks. Our method comprises two primary steps: first, we apply an automatic circuit-finding approach to identify the critical circuits within the model; second, we calculate the self-influence $I_H(x, x)$ for each layer of the circuit to assess the contribution of individual tokens to the model's decision-making process. By examining contributions at each layer, we can infer the thought process manifested by the model during reasoning. This layer-wise self-influence analysis provides a detailed understanding of the internal reasoning process while ensuring computational feasibility by focusing exclusively on the model's most essential components.

4.1 Automatic Circuit-Finding

To effectively locate the most important subgraphs, or circuits, in the model, we utilize advanced automatic circuit-finding methods, including Edge Attribution Patching (EAP), EAP-IG, and EAP-IG-KL. These methods allow us to identify and isolate the key circuits that contain essential features and their connecting weights, which are necessary for task completion. By focusing on these circuits rather than analyzing the entire model, we streamline the analysis and capture the primary information flow, thus attributing the model's output to specific components. This approach not only enhances efficiency but also enables us to focus on the most impactful areas of the model, laying a strong foundation for subsequent self-influence analysis.

In our approach, EAP identifies important edges by measuring the change in the loss function when each edge is perturbed, effectively constructing a "map" of the critical connections within the network. EAP-IG and EAP-IG-KL extend this by incorporating Integrated Gradients (IG) and Kullback-Leibler (KL) divergence, respectively. EAP-IG improves fidelity by more accurately capturing the importance of edges with low gradients, while EAP-IG-KL leverages KL divergence to ensure that the circuits faithfully represent the model's behavior under various interventions. Together, these methods allow us to efficiently locate the most faithful circuits in the network, ensuring that only the most relevant parts of the model are included in the analysis.

4.2 Self-Influence Calculation

Once critical circuit components have been isolated, the key remaining step is to interpret the computations performed by these components. Few methods have been proposed to interpret extracted circuits. Kevin Wang et al. (Wang et al., 2022) explored this by knocking out a single node—a (head, token position) pair in the circuit—revealing

heads with different functions. Arthur Conmy et al. (Conmy et al., 2023) proposed testing hypotheses about the functions implemented by each node in the circuit. Yunzhi Yao et al. (Yao et al., 2024) evaluated the impact of current knowledge editing techniques on these knowledge circuits, providing deeper insights into the functionality and limitations of these editing methodologies. We differ from these works by calculating the self-influence of each token in the sample across different layers of the circuit to reverse-engineer, which human-understandable reasoning patterns the model employs. We compute the self-influence $I_H(x, x)$ for each layer within the circuit. The self-influence formula is defined as:

$$I_H(x, x) = -\nabla_\theta L(x)^\top H^{-1} \nabla_\theta L(x) \quad (7)$$

where $\nabla_\theta L(x)$ is the gradient of the loss function $L(x)$ with respect to the parameters θ in the circuit at each layer, and x represents the tokens in the sample. The Hessian matrix H represents the second-order derivatives of the loss function with respect to the parameters, and H^{-1} is its inverse. Calculating self-influence allows us to measure each token's impact on the parameter updates, revealing the degree to which each token contributes to the decision-making process at each layer.

However, calculating the inverse of the Hessian H^{-1} directly is computationally expensive, particularly in large-scale models. To mitigate this, we adopt a divide-and-conquer strategy by utilizing the Hessian-vector product (HVP) in place of the explicit inverse calculation. The HVP approach allows us to approximate $H^{-1}v$ without direct inversion by first calculating $\frac{\partial f}{\partial x}(x)$, where x represents tokens in the sample, and then computing:

$$\frac{\partial x}{\partial} \left(\frac{\partial f(x)}{\partial x} \cdot v \right) \quad (8)$$

where $\frac{\partial f}{\partial x}(x) \in \mathbb{R}^{1\times d}$ represents the gradient of the function $f(x)$ with respect to the input tokens, and $v \in \mathbb{R}^{d\times 1}$ is a vector. This product $\left(\frac{\partial f(x)}{\partial x} \cdot v \right)$ is a scalar, and computing its gradient with respect to x is computationally efficient in deep learning frameworks like PyTorch and TensorFlow.

To further approximate $H^{-1}v$, we leverage a recursive Taylor expansion:

$$H^{-1} = \sum_{i=0}^{\infty} (I - H)^i \quad (9)$$

This expansion enables us to iteratively compute $H^{-1}v$, avoiding the computational expense of explicit inversion. Additionally, to ensure $||H|| \leq 1$, we scale the Hessian H by a factor $c \in \mathbb{R}_+$, allowing us to approximate H^{-1} as $c(cH)^{-1}$, which further reduces computational complexity. This approach enables efficient estimation of self-influence values, preserving the practicality of layer-wise influence analysis for large models.

With the self-influence values calculated across all layers within the circuit, we can now trace the flow of information and identify how different tokens contribute to the model's decision-making at each layer. By examining these contributions in a layer-by-layer fashion, we are able to infer the structure of the "reasoning tree" that the model implicitly follows during inference. This reasoning tree structure elucidates the hierarchical process by which the model accumulates and combines information, offering insights into the specific patterns of reasoning the model employs.

Our layer-wise self-influence analysis provides a comprehensive view of the internal mechanisms that drive the model's behavior. By focusing on critical components within the network, our method maintains computational feasibility while offering a fine-grained understanding of the model's decision-making process. This approach not only unveils the underlying reasoning patterns but also provides a valuable theoretical foundation for improving and optimizing transformer-based models for reasoning tasks.

As shown in Algorithm 1, this process of Circuit-Based Self-Influence Analysis allows us to construct the reasoning tree structure effectively.

5 Experiment

5.1 Experimental Setting

Dataset. We use the IOI dataset (Wang et al., 2022), designed to evaluate models' ability to perform indirect object identification tasks. Each entry consists of sentences with names and contexts, requiring the model to accurately predict the indirect object. The dataset contains minimal pairs of clean and corrupted inputs for direct comparison, testing the model's robustness in distinguishing between potential candidates, even when distractor names are introduced. For more dataset and metric details, see Appendix A.1.

Baselines. As SICAF is a mechanistic interpretation framework, we mainly implement it with previ-

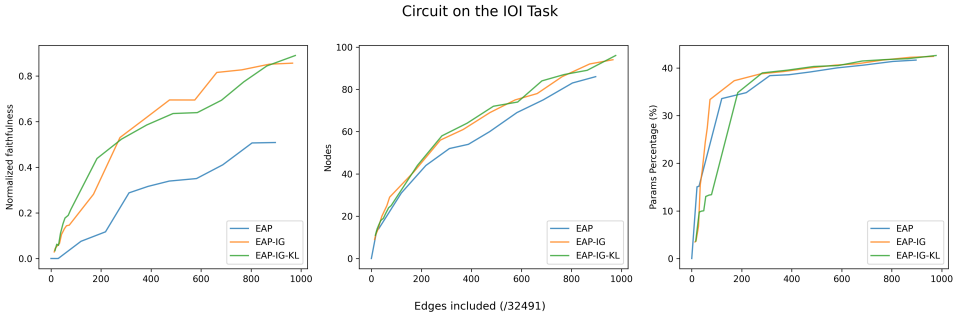

Figure 2: Comparison of normalized faithfulness, number of nodes, and parameter percentage for circuits identified by EAP, EAP-IG, and EAP-IG-KL on the IOI task. The x-axis represents the number of edges included, and each panel shows different metrics: normalized faithfulness (left), number of nodes (middle), and parameter percentage (right).

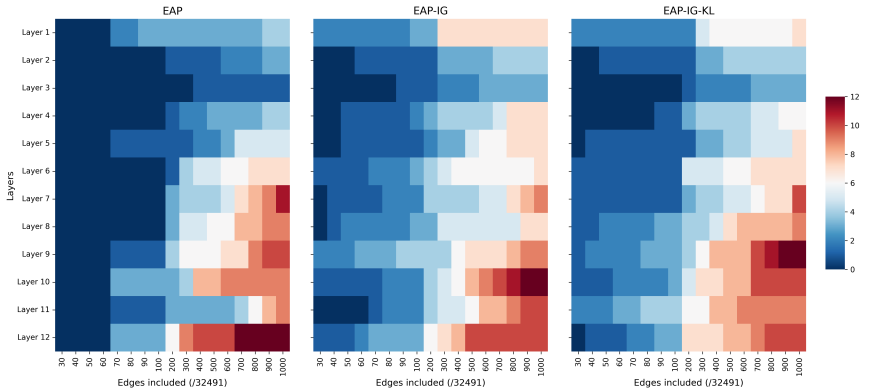

Figure 3: Heatmap of node importance across layers for EAP, EAP-IG, and EAP-IG-KL methods. The x-axis shows the number of edges included, and the y-axis shows the layers. Darker colors represent higher node importance, with EAP focusing on the last layers and EAP-IG, EAP-IG-KL showing more balanced distributions across layers.

ous Automatic Circuit-Finding approaches. Specifically, our baseline includes the following methods: EAP (Syed et al., 2023), which uses gradient-based approximations to estimate the importance of individual edges through their impact on model loss; EAP-IG (Hanna et al., 2024), which enhances EAP by employing integrated gradients along a path between clean and corrupted inputs to capture more accurate edge importance scores; and EAP-IG-KL (Hanna et al., 2024), which combines EAP-IG with Kullback-Leibler divergence as a loss function to generalize edge influence measurement across various interpretability tasks. Additional details are presented in Appendix A.3.

Implementation Details. We focus on simple tasks that are feasible even for GPT-2 small, which is the model most frequently studied from a circuits perspective. In our approach, we define GPT-2's attention heads and MLPs (multi-layer perceptrons) as nodes within its computational graph. Additional implementation details can be found in Appendix A.2.

5.2 Circuit Identification and Faithfulness

Using the EAP, EAP-IG, and EAP-IG-KL methods, we successfully identified small circuits (containing 1-2% of the edges) (see Table 1 for the circuit composition at 100 edges for each method, with additional results in Appendix B), recovering at least 85% of the model's performance on the IOI task. As shown in Figure 1 (left plot), both EAP-IG and EAP-IG-KL outperform EAP in terms of normalized faithfulness, achieving scores above 0.8 with approximately 800 edges, while EAP peaks below 0.6. This result suggests that EAP-IG and EAP-IG-KL are more effective at identifying circuits that retain model accuracy, particularly with fewer edges. Additionally, as seen in Figure 1 (center and right plots), EAP-IG and EAP-IG-KL exhibit rapid convergence in parameter inclusion, stabilizing around 35% of the total parameters to achieve high performance, while EAP requires close to 40% of the parameters to reach its maximum performance, which is still lower than the other methods. This result highlights the advantages of EAP-IG

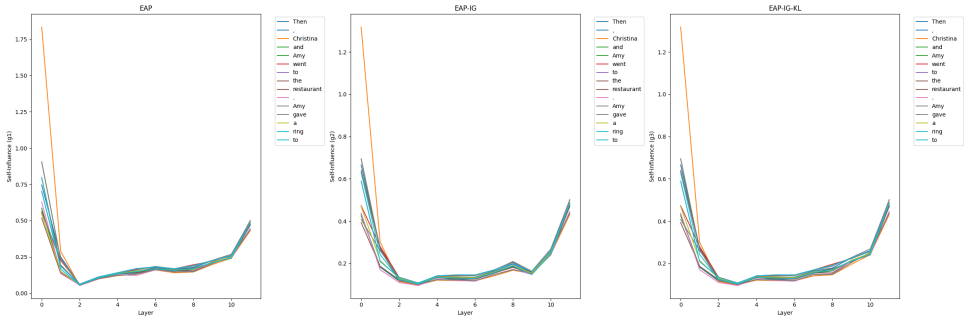

Figure 4: Self-influence scores of key tokens across model layers for the EAP, EAP-IG, and EAP-IG-KL methods on the IOI task. Each subplot represents the distribution of self-influence for individual tokens across the 12 layers of the GPT-2 model. EAP shows concentrated influence in the early and final layers, while EAP-IG and EAP-IG-KL display more balanced self-influence across layers, reflecting a structured progression of token importance. Key tokens such as "Christina," "Amy," and "gave" consistently show high self-influence, demonstrating their significance in the reasoning process.

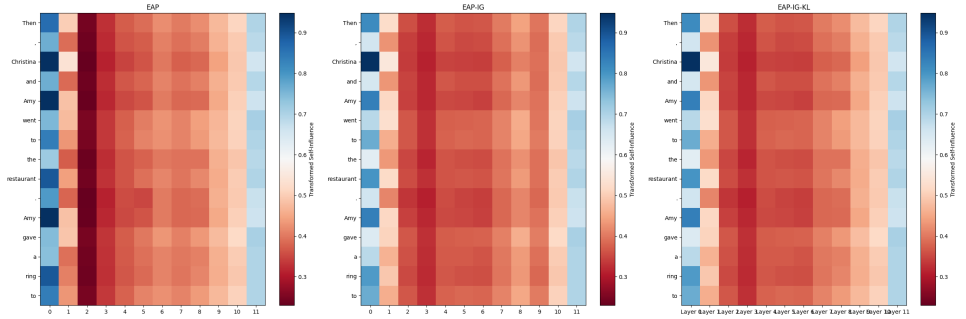

Figure 5: Heatmap of node importance across layers for EAP, EAP-IG, and EAP-IG-KL methods. The x-axis represents the layer indices, and the y-axis shows the tokens. Darker colors indicate higher node importance, with EAP focusing on the last layers and EAP-IG and EAP-IG-KL exhibiting a more balanced distribution across layers.

and EAP-IG-KL in identifying compact and faithful circuits, making them particularly suitable for tasks requiring high efficiency and accuracy.

Table 1: Circuit composition with 100 edges included using the greedy algorithm for EAP, EAP-IG, and EAP-IG-KL methods.

Method	**Circuit Composition**
EAP	input, a0.h1, a0.h10, m0, m4, a8.h10, a9.h3, a9.h4, m9, m10, a11.h2, a11.h3, m11, logits
EAP-IG	input, a0.h1, a0.h10, m0, m1, m2, a3.h0, m3, m4, a5.h5, a5.h9, m5, a6.h0, a6.h9, a7.h3, a7.h9, m7, a8.h3, a8.h6, a8.h10, m8, a9.h3, m9, a10.h2, m10, a11.h2, a11.h6, m11, logits
EAP-IG-KL	input, a0.h1, m0, m1, a3.h0, m4, a5.h5, a6.h9, a7.h1, a7.h3, a7.h9, a8.h6, a8.h10, m8, a9.h3, a9.h6, a9.h9, a10.h0, a10.h2, a10.h6, a10.h7, a11.h2, a11.h3, a11.h6, logits

5.3 Node Distribution Across Model Layers

As shown in Figure 2, the node distribution across model layers varies significantly between methods. EAP primarily activates nodes in the final layers (especially Layer 12), indicating that the model relies heavily on these layers to make final decisions in the IOI task. This pattern suggests that EAP emphasizes the semantic aggregation and decision generation occurring in the later layers. In contrast, EAP-IG and EAP-IG-KL demonstrate a more balanced node distribution across the early, middle, and final layers, with EAP-IG-KL showing substantial node activity between Layers 9 and 12. This pattern indicates that EAP-IG-KL captures more complex, multi-layered computational processes, utilizing information from a broader range of layers. Overall, the three methods share a consistent structure, with nodes concentrated in the first and last few layers, suggesting that these layers play a crucial role. The first layer likely helps in initial processing of input, while the last few layers appear critical for aggregating information before

Table 2: Token influence scores for Layer 0 across EAP, EAP-IG, and EAP-IG-KL methods.

Token	EAP	EAP-IG	EAP-IG-KL
Then	0.745	0.666	0.666
,	0.585	0.434	0.434
Christina	1.834	1.318	1.318
and	0.585	0.425	0.425
Amy	0.906	0.695	0.695
went	0.562	0.472	0.472
to	0.702	0.589	0.589
the	0.518	0.393	0.393
restaurant	0.797	0.639	0.639
.	0.628	0.437	0.437
Amy	0.906	0.695	0.695
gave	0.553	0.408	0.408

making final decisions.

5.4 The Model's Thought Process

Figures 3 and 4 provide insights into the layer-wise processing differences among the methods:EAP, EAP-IG, and EAP-IG-KL,specifically regarding their handling of self-influence scores. As shown in Figure 3, EAP concentrates self-influence scores predominantly in the final layers, indicating a focus on quickly aggregating high-level information. This approach makes EAP suitable for tasks where immediate decision-making is crucial, as it enables rapid synthesis of key contextual elements. However, this reliance on the final layers limits EAP's capability to capture intermediate reasoning steps, which are essential for handling more nuanced and multi-step reasoning tasks.

In contrast, Figure 4 illustrates that EAP-IG and EAP-IG-KL distribute influence scores more evenly across early, middle, and final layers, enabling a structured and gradual accumulation of information. This balanced distribution aligns well with tasks requiring multi-step reasoning, as it supports the retention and transformation of information throughout the model's layered structure. EAP-IG-KL, in particular, achieves a high level of balance, ensuring stable self-influence scores across layers. This feature suggests that EAP-IG-KL is not only more robust in handling complex reasoning tasks but also better equipped to leverage information from both lower and higher layers.

The common hierarchical reasoning path followed by all three methods is best illustrated using the sentence "Then, Christina and Amy went to the restaurant. Amy gave a ring to..." as an example. Here, in the early layers, key entities like "Christina" and "Amy" are identified, setting a foundational context that informs subsequent reasoning steps. Moving to the middle layers, the model interprets the action verb "gave," constructing relationships that frame "Amy" as the active entity in giving an item to another person. In the final layers, the model synthesizes this accumulated information, allowing it to conclude that "Amy" is the subject and infer the likely recipient.

The distinction among methods lies in the nuances of this shared reasoning path. EAP's emphasis on final-layer influence results in faster decision-making but may overlook subtler contextual nuances essential for complex inferences. On the other hand, EAP-IG distributes influence with a greater emphasis on intermediate layers, focusing on refining relational structures and contextual relationships as the reasoning pathway progresses. EAP-IG-KL exhibits the most balanced distribution, making it highly adaptable for tasks that require intricate relationships and consistent reasoning across all layers.

The detailed self-influence scores in Table 2 further support this analysis. For instance, in Layer 0, the token "Christina" exhibits a high self-influence score, reinforcing its role as a key contextual marker that informs initial reasoning. As the model advances, tokens such as "gave" and "Amy" demonstrate increased influence in the final layers, highlighting their relevance in constructing the concluding inference. Additional token influence scores across layers and for other samples are available in Appendix C, providing a comprehensive view of each method's impact across the reasoning process.

6 Conclusion

We propose a new mechanistic interpretation framework, SICAF, to trace and analyze the thought processes that language models (LMs) employ during complex reasoning tasks, and we validate our approach on the GPT-2 model in the IOI reasoning task. By applying circuit analysis and self-influence functions, we successfully mapped the reasoning pathways within the GPT-2 model during the IOI task. Our method reveals a hierarchical structure in the model's reasoning process, distinguishing key entities and relationships in a manner that resembles human reasoning steps. Overall, our findings contribute to a more interpretable and sys-

tematic understanding of the reasoning processes in language models, enhancing the transparency and trustworthiness of AI systems.

7 Limitations

While our study successfully elucidates certain thought processes within language models, it has some limitations. First, the analysis was conducted primarily on the GPT-2 model and may not generalize to larger or different architectures without adaptation. Additionally, calculating self-influence requires computationally intensive methods, which may pose scalability challenges for more complex models. Finally, our work focused on a single task (Indirect Object Identification, or IOI), and the applicability of these findings to other natural language processing tasks remains an open question. Future research should explore the adaptability of this approach across varied tasks and model architectures, as well as investigate methods to optimize computational efficiency.

Acknowledgement

Di Wang and Lijie Hu are supported in part by the funding BAS/1/1689-01-01, URF/1/4663-01-01, REI/1/5232-01-01, REI/1/5332-01-01, and URF/1/5508-01-01 from KAUST, and funding from KAUST - Center of Excellence for Generative AI, under award number 5940.

References

Naman Agarwal, Brian Bullins, and Elad Hazan. 2017. Second-order stochastic optimization for machine learning in linear time. *Journal of Machine Learning Research*, 18(116):1–40.

S Basu, P Pope, and S Feizi. 2021. Influence functions in deep learning are fragile. In *International Conference on Learning Representations (ICLR)*.

Yonatan Belinkov. 2022. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219.

L Bereska and E Gavves. 2024. Mechanistic interpretability for ai safety–a review. *arXiv preprint arXiv:2404.14082*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Marc-Etienne Brunet, Colleen Alkalay-Houlihan, Ashton Anderson, and Richard Zemel. 2019. Understanding the origins of bias in word embeddings. In *International conference on machine learning*, pages 803–811. PMLR.

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

Giuseppe Casalicchio, Christoph Molnar, and Bernd Bischl. 2019. Visualizing the feature importance for black box models. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part I*, pages 655–670. Springer International Publishing.

Hongge Chen, Si Si, Yang Li, Ciprian Chelba, Sanjiv Kumar, Duane Boning, and Cho-Jui Hsieh. 2020. Multi-stage influence function. *Advances in Neural Information Processing Systems*, 33:12732–12742.

Zeming Chen, Gail Weiss, Eric Mitchell, Asli Celikyilmaz, and Antoine Bosselut. 2023. Reckoning: Reasoning through dynamic knowledge encoding. *CoRR*, abs/2305.06349.

Keyuan Cheng, Gang Lin, Haoyang Fei, Lu Yu, Muhammad Asif Ali, Lijie Hu, Di Wang, et al. 2024. Multi-hop question answering under temporal knowledge editing. *arXiv preprint arXiv:2404.00492*.

Aidan Conmy, Alex Mavor-Parker, Anthony Lynch, et al. 2023. Towards automated circuit discovery for mechanistic interpretability. *Advances in Neural Information Processing Systems*, 36:16318–16352.

R Dennis Cook. 2000. Detection of influential observation in linear regression. *Technometrics*, 42(1):65–68.

R Dennis Cook and Sanford Weisberg. 1980. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics*, 22(4):495–508.

Ian Covert, Scott Lundberg, and Su-In Lee. 2021. Explaining by removing: A unified framework for model explanation. *Journal of Machine Learning Research*, 22(209):1–90.

Antonia Creswell and Murray Shanahan. 2022. Faithful reasoning using large language models. *CoRR*, abs/2208.14271.

Antonia Creswell, Murray Shanahan, and Irina Higgins. 2022. Selection-inference: Exploiting large language models for interpretable logical reasoning. *CoRR*, abs/2205.09712.

Yue Dong, Chandra Bhagavatula, Ximing Lu, Jena D. Hwang, Antoine Bosselut, Jackie Chi Kit Cheung, and Yejin Choi. 2021. On-the-fly attention modulation for neural generation. In *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume Findings of ACL, pages 1261–1274. Association for Computational Linguistics.

Andreas Geiger, Hongjing Lu, Thomas Icard, et al. 2021. Causal abstractions of neural networks. *Advances in Neural Information Processing Systems*, 34:9574–9586.

Mor Geva, Roei Schuster, Jonathan Berant, et al. 2020. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*.

Aditya Golatkar, Alessandro Achille, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. 2021. Mixed-privacy forgetting in deep networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 792–801.

Aditya Golatkar, Alessandro Achille, and Stefano Soatto. 2020. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9304–9312.

Han Guo, Nazneen Rajani, Peter Hase, Mohit Bansal, and Caiming Xiong. 2021. Fastif: Scalable influence functions for efficient model interpretation and debugging. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10333–10350.

Xiaochuang Han, Byron C Wallace, and Yulia Tsvetkov. 2020. Explaining black box predictions and unveiling data artifacts through influence functions. *arXiv preprint arXiv:2005.06676*.

Moya Hanna, Sandro Pezzelle, and Yonatan Belinkov. 2024. Have faith in faithfulness: Going beyond circuit overlap when finding model mechanisms. *arXiv preprint arXiv:2403.17806*.

Zhaozhi He, Xinyuan Ge, Qixun Tang, et al. 2024. Dictionary learning improves patch-free circuit discovery in mechanistic interpretability: A case study on othello-gpt. *arXiv preprint arXiv:2402.12201*.

Yihuai Hong, Yuelin Zou, Lijie Hu, Ziqian Zeng, Di Wang, and Haiqin Yang. 2024. Dissecting fine-tuning unlearning in large language models. *arXiv preprint arXiv:2410.06606*.

Y. Hou, J. Li, Y. Fei, et al. 2023. Towards a mechanistic interpretation of multi-step reasoning capabilities of language models. *arXiv preprint arXiv:2310.14491*.

Lijie Hu, Liang Liu, Shu Yang, Xin Chen, Hongru Xiao, Mengdi Li, Pan Zhou, Muhammad Asif Ali, and Di Wang. 2024a. A hopfieldian view-based interpretation for chain-of-thought reasoning. *arXiv preprint arXiv:2406.12255*.

Lijie Hu, Yixin Liu, Ninghao Liu, Mengdi Huai, Lichao Sun, and Di Wang. Improving interpretation faithfulness for vision transformers. In *Forty-first International Conference on Machine Learning*.

Lijie Hu, Chenyang Ren, Zhengyu Hu, Hongbin Lin, Cheng-Long Wang, Hui Xiong, Jingfeng Zhang, and Di Wang. 2024b. Editable concept bottleneck models. *arXiv preprint arXiv:2405.15476*.

Lijie Hu, Chenyang Ren, Huanyi Xie, Khouloud Saadi, Shu Yang, Jingfeng Zhang, and Di Wang. 2024c. Dissecting misalignment of multimodal large language models via influence function. *arXiv preprint arXiv:2411.11667*.

Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Yongchan Kwon, Eric Wu, Kevin Wu, and James Zou. 2023. Datainf: Efficiently estimating data influence in lora-tuned llms and diffusion models. In *The Twelfth International Conference on Learning Representations*.

Jiaqi Liu, Jian Lou, Zhan Qin, and Kui Ren. 2024. Certified minimax unlearning with generalization rates and deletion capacity. *Advances in Neural Information Processing Systems*, 36.

Kevin Meng, David Bau, Alex Andonian, et al. 2022. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372.

Edward Michaud, Ziming Liu, Ugur Girit, et al. 2024. The quantization model of neural scaling. *Advances in Neural Information Processing Systems*, 36.

John Miller, Bilal Chughtai, and William Saunders. 2024. Transformer circuit faithfulness metrics are not robust. *arXiv preprint arXiv:2407.08734*.

Neel Nanda. 2023. Mechanistic interpretability quickstart guide. Neel Nanda's Blog. Accessed: 2023-01-26.

Chris Olah. 2022. Mechanistic interpretability, variables, and the importance of interpretable bases. https://www.transformer-circuits.pub/2022/mech-interp-essay.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. 2020. Zoom in: An introduction to circuits. *Distill*.

Alec Radford, Jeffrey Wu, Rewon Child, et al. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, pages 3145–3153. PMLR.

Daniel Smilkov, Nikhil Thorat, Been Kim, et al. 2017. Smoothgrad: Removing noise by adding noise. *arXiv preprint arXiv:1706.03825*.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, pages 3319–3328. PMLR.

Aaquib Syed, Charles Rager, and Aidan Conmy. 2023. Attribution patching outperforms automated circuit discovery. *arXiv preprint arXiv:2310.10348*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008.

Hao Wang, Berk Ustun, and Flavio Calmon. 2019. Repairing without retraining: Avoiding disparate impact with counterfactual distributions. In *International Conference on Machine Learning*, pages 6618–6627. PMLR.

Kai Wang, Anna Variengien, Aidan Conmy, et al. 2022. Interpretability in the wild: A circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*.

Alexander Warnecke, Lukas Pirch, Christian Wressnegger, and Konrad Rieck. 2023. Machine unlearning of features and labels. *Network and Distributed System Security (NDSS) Symposium*.

Alex Warstadt, Alicia Parrish, Haokun Liu, et al. 2020. Blimp: The benchmark of linguistic minimal pairs for english. *Transactions of the Association for Computational Linguistics*, 8:377–392.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Shu Yang, Muhammad Asif Ali, Cheng-Long Wang, Lijie Hu, and Di Wang. 2024a. Moral: Moe augmented lora for llms' lifelong learning. *arXiv preprint arXiv:2402.11260*.

Shu Yang, Shenzhe Zhu, Ruoxuan Bao, Liang Liu, Yu Cheng, Lijie Hu, Mengdi Li, and Di Wang. 2024b. What makes your model a low-empathy or warmth person: Exploring the origins of personality in llms. *arXiv preprint arXiv:2410.10863*.

Yunzhi Yao, Ning Zhang, Zhihao Xi, et al. 2024. Knowledge circuits in pretrained transformers. *arXiv preprint arXiv:2405.17969*.

Zhuoran Zhang, Yongxiang Li, Zijian Kan, Keyuan Cheng, Lijie Hu, and Di Wang. 2024. Locate-then-edit for multi-hop factual recall under knowledge editing. *arXiv preprint arXiv:2410.06331*.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. 2023. Representation engineering: A top-down approach to ai transparency. *CoRR*.

A Additional Experiments

A.1 Datasets

The IOI (Indirect Object Identification) dataset (Wang et al., 2023) is designed to evaluate a model's ability to identify indirect objects within sentences. This task requires the model to recognize the indirect object among two specific names in a given sentence, predicting which name serves as the indirect object of the sentence. The dataset includes both clean and corrupted inputs. In clean inputs, each entry consists of a pair of names with contextual information, prompting the model to accurately identify the indirect object, or recipient, of an action. For example, in the sentence "When Amy and Laura got a snack at the house, Laura decided to give it to," the model should predict "Amy" as the indirect object. In corrupted inputs, the sentence structure remains unchanged, but the original indirect object is replaced by a third name, making both "Amy" and "Laura" approximately equally probable. For instance, replacing "Laura" with "Nicholas" yields "When Amy and Laura got a snack at the house, Nicholas decided to give it to," or replacing "Laura" with "Amy" results in "When Amy and Laura got a snack at the house, Amy decided to give it to," increasing the model's difficulty in distinguishing the correct indirect object. The model's predictions are evaluated using logit difference (logit diff), calculated as the logit of the target indirect object minus the logit of the distractor. A larger logit difference indicates a stronger preference by the model for the correct indirect object. Using Wang et al.'s data generator, we generated a dataset of 1000 sentences table3,

containing both clean and corrupted inputs, for experimental analysis of the model's performance on this task.

A.2 Details of Experimental Settings

For the IOI task, we used a corpus of 1000 sentences created by (Hanna et al., 2024), based on the dataset generator from (Wang et al., 2022), which includes both clean and corrupted inputs. We first fine-tuned GPT-2 on this dataset and then followed the experimental setup outlined in Michael Hanna's paper, defining GPT-2's attention heads and MLPs (multi-layer perceptrons) as nodes in its computational graph. A node's output directed to another node is defined as an edge. The number of nodes was 158, and the number of edges was 32,491. The input to node v is the sum of the outputs from all nodes u connected to v. For $n = 30, 40, \ldots, 100, 200, \ldots, 1000$, we selected a circuit of n edges using a greedy search procedure. Generally, larger n results in more faithful circuits, but these circuits tend to be less localized and interpretable. Let b and b' represent the model's performance on clean and corrupted inputs, respectively. The circuit's performance and faithfulness m are normalized as $(m - b')/(b - b')$.

A.3 Baselines

EAP: EAP (Edge Attribution Patching) (Syed et al., 2023), which uses gradient-based approximations to assess the importance of individual edges by estimating the loss change from intervening on each model edge, allowing for a scalable approximation of causal effects without requiring extensive forward passes.

EAP-IG: EAP-IG (EAP with Integrated Gradients) (Hanna et al., 2024), which improves upon EAP by employing integrated gradients along a path between clean and corrupted inputs to compute a more faithful edge importance score, capturing a wider range of influence and avoiding zero-gradient issues common in standard gradient calculations.

EAP-IG-KL: EAP-IG-KL (Hanna et al., 2024), which combines EAP-IG with Kullback-Leibler (KL) divergence as a loss function, enabling it to generalize across tasks by measuring divergence between the patched and original model outputs, thus allowing consistent applicability to various interpretability tasks.

The table below (Table 4) provides a comparison of the EAP, EAP-IG, and EAP-IG-KL methods, highlighting their distinctive attributes in terms of granularity, component, value, token positions, direction, and set used in the circuit-finding approach:

B Circuit Results

"edges included": 200,
"greedy algorithm":
EAP: input, a0.h1, a0.h10, m0, m1, m3, m4, m5, a6.h0, a6.h9, m6, a7.h3, a7.h9, m7, a8.h3, a8.h6, a8.h10, m8, a9.h3, a9.h4, m9, a10.h2, a10.h7, m10, a11.h1, a11.h2, a11.h3, a11.h6, a11.h8, m11, logits
EAP-IG: input, a0.h1, a0.h10, m0, m1, m2, a3.h0, a3.h4, m3, a4.h3, m4, a5.h1, a5.h5, a5.h9, m5, a6.h0, a6.h6, a6.h9, m6, a7.h3, a7.h9, m7, a8.h3, a8.h6, a8.h10, m8, a9.h3, a9.h4, a9.h6, m9, a10.h0, a10.h2, a10.h7, m10, a11.h1, a11.h2, a11.h3, a11.h6, a11.h9, m11, logits
EAP-IG-KL: input, a0.h1, m0, m1, m2, a3.h0, a3.h4, m3, m4, a5.h1, a5.h5, a5.h8, a5.h9, m5, a6.h0, a6.h6, a6.h9, a7.h1, a7.h3, a7.h9, m7, a8.h3, a8.h6, a8.h10, m8, a9.h3, a9.h4, a9.h6, a9.h9, m9, a10.h0, a10.h1, a10.h2, a10.h6, a10.h7, m10, a11.h1, a11.h2, a11.h3, a11.h6, a11.h9, a11.h10, m11, logits

"edges included": 300,
"greedy algorithm":
EAP: input, a0.h1, a0.h10, m0, m1, a3.h0, m3, m4, a5.h1, a5.h8, a5.h9, m5, a6.h0, a6.h6, a6.h9, m6, a7.h1, a7.h3, a7.h9, a7.h11, m7, a8.h3, a8.h5, a8.h6, a8.h10, a8.h11, m8, a9.h3, a9.h4, a9.h8, m9, a10.h2, a10.h7, m10, a11.h0, a11.h1, a11.h2, a11.h3, a11.h6, a11.h8, a11.h9, a11.h11, m11, logits
EAP-IG: input, a0.h1, a0.h3, a0.h4, a0.h5, a0.h9, a0.h10, m0, a1.h0, a1.h11, m1, a2.h2, m2, a3.h0, a3.h4, a3.h6, m3, a4.h3, a4.h4, m4, a5.h1, a5.h5, a5.h9, a5.h10, m5, a6.h0, a6.h6, a6.h9, m6, a7.h1, a7.h3, a7.h9, a7.h11, m7, a8.h3, a8.h6, a8.h10, m8, a9.h3, a9.h4, a9.h6, a9.h9, m9, a10.h0, a10.h2, a10.h6, a10.h7, m10, a11.h1, a11.h2, a11.h3, a11.h6, a11.h8, a11.h9, m11, logits
EAP-IG-KL: input, a0.h1, a0.h3, a0.h4, a0.h5, m0, a1.h0, a1.h11, m1, a2.h2, m2, a3.h0, a3.h4, a3.h6, m3, a4.h3, a4.h4, m4, a5.h1, a5.h5, a5.h8, a5.h9, m5, a6.h0, a6.h6, a6.h9, m6, a7.h1, a7.h3, a7.h7, a7.h9, m7, a8.h1, a8.h3, a8.h5, a8.h6, a8.h8, a8.h10, a8.h11, m8, a9.h2, a9.h3, a9.h4, a9.h6, a9.h7, a9.h9, m9, a10.h0, a10.h1, a10.h2, a10.h6, a10.h7, a10.h10, a10.h11, m10, a11.h1, a11.h2, a11.h3, a11.h6, a11.h9, a11.h10, m11, logits

"edges included": 400,
"greedy algorithm":
EAP: input, a0.h1, a0.h10, m0, m1, m2, a3.h0, m3, a4.h4, m4, a5.h1, a5.h5, a5.h8, a5.h9, m5, a6.h0, a6.h6, a6.h9, m6, a7.h1, a7.h3, a7.h9, a7.h11, m7, a8.h3, a8.h5, a8.h6, a8.h10, a8.h11, m8, a9.h2, a9.h3, a9.h4, a9.h5, a9.h7, a9.h8, a9.h11, m9, a10.h2, a10.h7, m10, a11.h0, a11.h1, a11.h2, a11.h3, a11.h6, a11.h8, a11.h9, a11.h10, a11.h11, m11, logits
EAP-IG: input, a0.h1, a0.h3, a0.h4, a0.h5, a0.h9, a0.h10, m0, a1.h0, a1.h11, m1, a2.h2, m2, a3.h0, a3.h4, a3.h6, m3, a4.h3, a4.h4, m4, a5.h1, a5.h5, a5.h8, a5.h9, a5.h10, m5, a6.h0, a6.h6, a6.h9, m6, a7.h1, a7.h3, a7.h9, a7.h11, m7, a8.h3, a8.h5, a8.h6, a8.h10, a8.h11, m8, a9.h3, a9.h4, a9.h6, a9.h7, a9.h9, m9, a10.h0, a10.h2, a10.h6, a10.h7, m10, a11.h1, a11.h2, a11.h3, a11.h6, a11.h8, a11.h9, a11.h10, m11, logits
EAP-IG-KL: input, a0.h1, a0.h3, a0.h4, a0.h5, a0.h10, m0, a1.h0, a1.h11, m1, a2.h2, m2, a3.h0, a3.h4, a3.h6, m3, a4.h3, a4.h4, m4, a5.h1, a5.h5, a5.h8, a5.h9, m5, a6.h0, a6.h6, a6.h9, m6, a7.h1, a7.h3, a7.h6, a7.h7, a7.h9, a7.h11, m7, a8.h1, a8.h3, a8.h5, a8.h

"edges included": 500,
"greedy algorithm":
EAP: input, a0.h1, a0.h10, m0, m1, m2, a3.h0, a3.h6, m3, a4.h4, m4, a5.h1, a5.h5, a5.h8, a5.h9, m5, a6.h0, a6.h6, a6.h9, m6, a7.h1, a7.h3, a7.h6, a7.h9, a7.h11, m7, a8.h3, a8.h5, a8.h6, a8.h10, a8.h11, m8, a9.h2, a9.h3, a9.h4, a9.h5, a9.h7, a9.h8, a9.h11, m9, a10.h2, a10.h7, m10, a11.h0, a11.h1, a11.h2, a11.h3, a11.h6, a11.h8, a11.h9, a11.h10, a11.h11, m11, logits
EAP-IG: input, a0.h1, a0.h3, a0.h4, a0.h5, a0.h9, a0.h10, m0, a1.h0, a1.h11, m1, a2.h2, m2, a3.h0, a3.h4, a3.h6, m3, a4.h3, a4.h4, a4.h6, a4.h7, m4, a5.h1, a5.h5, a5.h8, a5.h9, a5.h10, m5, a6.h0, a6.h6, a6.h9, m6, a7.h1, a7.h3, a7.h9, a7.h11, m7, a8.h1, a8.h3, a8.h5, a8.h6, a8.h10, a8.h11, m8, a9.h2, a9.h3, a9.h4, a9.h6, a9.h7, a9.h8, a9.h9, m9, a10.h0, a10.h2, a10.h6, a10.h7, a10.h10, m10, a11.h0, a11.h1, a11.h2, a11.h3, a11.h6, a11.h8, a11.h9, a11.h10, a11.h11, m11, logits
EAP-IG-KL: input, a0.h1, a0.h3, a0.h4, a0.h5, a0.h10, m0, a1.h0, a1.h5, a1.h11, m1, a2.h2, m2, a3.h0, a3.h4, a3.h6, m3, a4.h3, a4.h4, a4.h6, m4, a5.h1, a5.h5, a5.h8, a5.h9, a5.h10, m5, a6.h0, a6.h6, a6.h7, a6.h9, m6, a7.h1, a7.h3, a7.h6, a7.h7, a7.h9, a7.h11, m7, a8.h1, a8.h3, a8.h5, a8.h6, a8.h8, a8.h10, a8.h11, m8, a9.h0, a9.h2, a9.h3, a9.h4, a9.h6, a9.h7, a9.h9, m9, a10.h0, a10.h1, a10.h2, a10.h6, a10.h7, a10.h10, a10.h11, m10, a11.h1, a11.h2, a11.h3, a11.h6, a11.h8, a11.h9, a11.h10, m11, logits

(Results for edge values between 500 and 1000 follow similar patterns and are not listed in full here.)

"edges included": 1000,
"greedy algorithm":
EAP: input, a0.h1, a0.h9, a0.h10, m0, a1.h8, a1.h11, m1, m2, a3.h0, a3.h6, a3.h10, m3, a4.h3, a4.h4, a4.h6, a4.h7, m4, a5.h0, a5.h1, a5.h5, a5.h8, a5.h9, a5.h10, m5, a6.h0, a6.h1, a6.h2, a6.h3, a6.h4, a6.h5, a6.h6, a6.h8, a6.h9, a6.h10, m6, a7.h1, a7.h3, a7.h5, a7.h6, a7.h7, a7.h9, a7.h10, a7.h11, m7, a8.h0, a8.h1, a8.h3, a8.h5, a8.h6, a8.h7, a8.h9, a8.h10, a8.h11, m8, a9.h1, a9.h2, a9.h3, a9.h4, a9.h5, a9.h7, a9.h8, a9.h11, m9, a10.h0, a10.h1, a10.h2, a10.h3, a10.h4, a10.h7, a10.h8, a10.h10, m10, a11.h0, a11.h1, a11.h2, a11.h3, a11.h4, a11.h5, a11.h6, a11.h8, a11.h9, a11.h10, a11.h11, m11, logits
EAP-IG: input, a0.h1, a0.h3, a0.h4, a0.h5, a0.h9, a0.h10, m0, a1.h0, a1.h5, a1.h11, m1, a2.h2, a2.h9, m2, a3.h0, a3.h3, a3.h4, a3.h6, a3.h7, a3.h10, m3, a4.h3, a4.h4, a4.h6, a4.h7, a4.h8, a4.h11, m4, a5.h0, a5.h1, a5.h5, a5.h8, a5.h9, a5.h10, m5, a6.h0, a6.h1, a6.h4, a6.h5, a6.h6, a6.h7, a6.h8, a6.h9, m6, a7.h1, a7.h3, a7.h5, a7.h7, a7.h9 EAP-IG-KL: input, a0.h1, a0.h3, a0.h4, a0.h5, a0.h7, a0.h10, m0, a1.h0, a1.h5, a1.h11, m1, a2.h2, a2.h9, m2, a3.h0, a3.h3, a3.h4, a3.h6, a3.h7, m3, a4.h3, a4.h4, a4.h5, a4.h6, a4.h7, a4.h8, m4, a5.h1, a5.h5, a5.h8, a5.h9, a5.h10, a5.h11, m5, a6.h0, a6.h1, a6.h3, a6.h5, a6.h6, a6.h7, a6.h8, a6.h9, a6.h10, m6, a7.h1, a7.h3, a7.h5, a7.h6, a7.h7, a7.h8, a7.h9, a7.h11, m7, a8.h0, a8.h1, a8.h2, a8.h3, a8.h5, a8.h6, a8.h7, a8.h8, a8.h9, a8.h10, a8.h11, m8, a9.h0, a9.h2, a9.h3, a9.h4, a9.h5, a9.h6, a9.h7, a9.h9, a9.h11, m9, a10.h0, a10.h1, a10.h2, a10.h3, a10.h6, a10.h7, a10.h10, a10.h11, m10, a11.h0, a11.h1, a11.h2, a11.h3, a11.h6, a11.h8, a11.h9, a11.h10, a11.h11, m11, logits

C Token self-Influence

See Tables 5, 6, and 7 for complete results. Other results are similar and are not displayed.

Algorithm 1: Self-Influence Circuit Analysis Framework

Input: Model M, Input sample x, Circuit-finding methods $\{\text{EAP}, \text{EAP-IG}, \text{EAP-IG-KL}\}$, Scaling factor c
Output: Inferred thought process structure
Initialize model parameters θ;
Select circuit-finding method (e.g., EAP, EAP-IG, or EAP-IG-KL);
Phase 1: Automatic Circuit Identification
Function `IdentifyCriticalCircuits(`M, x, *method*`)`**:**
  for *each edge* (u, v) *in* M **do**
    Perturb edge (u, v) to evaluate effect on loss $L(x)$;
    Calculate edge importance score S_{uv} based on perturbation:

$$S_{uv} = \Delta L(x; u, v)$$

  Rank edges by importance and select top-k;
  return CircuitGraph (subgraph with highest-ranked edges);
CircuitGraph $\leftarrow$ `IdentifyCriticalCircuits(`M, x, *chosen method*`)`;
Phase 2: Layer-wise Self-Influence Computation
Function `ComputeSelfInfluence(`*CircuitGraph*, x, θ, c`)`**:**
  for *each layer* ℓ *in CircuitGraph* **do**
    Initialize self-influence $I_\ell(x, x) \leftarrow 0$;
    Compute gradient $\nabla_\theta L(x)$ w.r.t. parameters θ at layer ℓ;
    Calculate Hessian matrix H based on $\nabla_\theta L(x)$;
    if $||H|| > 1$ **then**
      Scale Hessian by factor c: $H \leftarrow cH$;
    Approximate H^{-1} using Taylor expansion:

$$H^{-1} \approx \sum_{i=0}^{\infty} (I - H)^i$$

    Calculate self-influence for layer ℓ:

$$I_\ell(x, x) = -\nabla_\theta L(x)^\top H^{-1} \nabla_\theta L(x)$$

  return LayerwiseInfluenceScores for all layers;
LayerwiseInfluenceScores $\leftarrow$ `ComputeSelfInfluence(`*CircuitGraph*, x, θ, c`)`;
Phase 3: Thought Process Inference
Function `InferThoughtProcess(`*LayerwiseInfluenceScores*`)`**:**
  Initialize ThoughtProcess $\leftarrow \{\}$;
  for *each layer* ℓ **do**
    Analyze distribution $\{I_\ell(x, x_i)\}_{i=1}^n$ across tokens;
    Identify significant contributions to model's decision pathway at layer ℓ;
    Update ThoughtProcess with token importance at each layer;
  return ThoughtProcess;
ThoughtProcess $\leftarrow$ `InferThoughtProcess(`*LayerwiseInfluenceScores*`)`;

return ThoughtProcess;

Table 3: Sample entries from the IOI (Indirect Object Identification) dataset, showcasing clean, corrupted, and corrupted hard inputs along with target and distractor indices.

Clean Input		
Sentence	**Target Index**	**Distractor Index**
When Amy and Laura got a snack at the house, Laura decided to give it to	14235	16753
Then, Danielle and Andrew had a lot of fun at the office. Andrew gave a computer to	39808	6858
When Anthony and Jose got a drink at the restaurant, Jose decided to give it to	9953	5264
Then, Sean and Vanessa had a long argument, and afterwards Vanessa said to	11465	42100
.....		
Corrupted Input		
Sentence	**Target Index**	**Distractor Index**
When Amy and Laura got a snack at the house, Nicholas decided to give it to	14235	16753
Then, Danielle and Andrew had a lot of fun at the office. Jeremy gave a computer to	39808	6858
When Anthony and Jose got a drink at the restaurant, Nathan decided to give it to	9953	5264
Then, Sean and Vanessa had a long argument, and afterwards Kimberly said to	11465	42100
.....		
Corrupted Hard Input		
Sentence	**Target Index**	**Distractor Index**
When Amy and Laura got a snack at the house, Amy decided to give it to	14235	16753
Then, Danielle and Andrew had a lot of fun at the office. Danielle gave a computer to	39808	6858
When Anthony and Jose got a drink at the restaurant, Anthony decided to give it to	9953	5264
Then, Sean and Vanessa had a long argument, and afterwards Sean said to	11465	42100
.....		

Table 4: Comparison of EAP, EAP-IG, and EAP-IG-KL methods. Each method differs in at least one aspect in terms of granularity, component, value, token positions, direction, and set.

Method	**Granularity**	**Component**	**Value**	**Token Positions**	**Direction**	**Set**
EAP	Edges	Edge	Resample	All tokens	Resample Clean	Circuit
EAP-IG	Edges	Edge	Integrated Gradient Path	Specific tokens	Resample Clean	Circuit
EAP-IG-KL	Edges	Edge	Integrated Gradient + KL Divergence	Specific tokens	Resample Clean	Circuit

Table 5: Self-influence scores across layers 0–11 using three different methods. The best influence score within each layer is highlighted in bold.

Token	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
EAP												
Then	0.745	0.251	0.059	0.107	0.138	0.150	0.178	0.163	0.178	0.224	0.261	0.484
,	0.585	0.150	0.056	0.103	0.132	0.139	0.166	0.153	0.164	0.214	0.246	0.469
Christina	1.834	0.289	0.052	0.098	0.120	0.132	0.160	0.141	0.146	0.197	0.242	0.431
and	0.585	0.148	0.057	0.104	0.133	0.143	0.168	0.155	0.167	0.213	0.246	0.476
Amy	0.906	0.233	0.055	0.101	0.123	0.132	0.163	0.147	0.149	0.205	0.252	0.438
went	0.562	0.226	0.062	0.112	0.139	0.165	0.183	0.168	0.196	0.221	0.258	0.501
to	0.702	0.186	0.061	0.112	0.141	0.170	0.181	0.169	0.190	0.224	0.257	0.486
the	0.518	0.138	0.056	0.104	0.132	0.142	0.161	0.154	0.155	0.212	0.240	0.472
restaurant	0.797	0.194	0.058	0.109	0.132	0.152	0.172	0.157	0.174	0.208	0.247	0.486
.	0.628	0.145	0.053	0.098	0.126	0.123	0.161	0.146	0.150	0.213	0.244	0.446
Amy	0.906	0.233	0.055	0.101	0.123	0.132	0.163	0.147	0.149	0.205	0.252	0.438
gave	0.553	0.240	0.062	0.110	0.137	0.160	0.185	0.167	0.187	0.227	0.268	0.499
EAP-IG												
Then	0.666	0.275	0.129	0.104	0.138	0.133	0.134	0.163	0.197	0.157	0.261	0.484
,	0.434	0.182	0.117	0.100	0.132	0.124	0.125	0.153	0.183	0.150	0.246	0.469
Christina	1.318	0.300	0.114	0.099	0.120	0.118	0.117	0.141	0.167	0.153	0.242	0.431
and	0.425	0.185	0.120	0.101	0.133	0.128	0.128	0.155	0.186	0.151	0.246	0.476
Amy	0.695	0.266	0.119	0.100	0.123	0.122	0.118	0.147	0.170	0.155	0.252	0.438
went	0.472	0.279	0.135	0.107	0.139	0.143	0.145	0.168	0.207	0.157	0.258	0.501
to	0.589	0.209	0.133	0.107	0.141	0.145	0.145	0.169	0.201	0.161	0.257	0.486
the	0.393	0.187	0.122	0.100	0.132	0.128	0.125	0.154	0.181	0.152	0.240	0.472
restaurant	0.639	0.271	0.125	0.106	0.132	0.134	0.133	0.157	0.190	0.152	0.247	0.486
.	0.437	0.170	0.110	0.095	0.126	0.119	0.116	0.146	0.172	0.147	0.244	0.446
Amy	0.695	0.266	0.119	0.100	0.123	0.122	0.118	0.147	0.170	0.155	0.252	0.438
gave	0.408	0.260	0.134	0.107	0.137	0.140	0.142	0.167	0.209	0.163	0.268	0.499
EAP-IG-KL												
Then	0.666	0.275	0.129	0.104	0.138	0.133	0.134	0.163	0.178	0.225	0.261	0.484
,	0.434	0.182	0.117	0.100	0.132	0.124	0.125	0.153	0.164	0.214	0.246	0.469
Christina	1.318	0.300	0.114	0.099	0.120	0.118	0.117	0.141	0.146	0.195	0.242	0.431
and	0.425	0.185	0.120	0.101	0.133	0.128	0.128	0.155	0.167	0.213	0.246	0.476
Amy	0.695	0.266	0.119	0.100	0.123	0.122	0.118	0.147	0.149	0.205	0.252	0.438
went	0.472	0.279	0.135	0.107	0.139	0.143	0.145	0.168	0.196	0.222	0.258	0.501
to	0.589	0.209	0.133	0.107	0.141	0.145	0.145	0.169	0.190	0.224	0.257	0.486
the	0.393	0.187	0.122	0.100	0.132	0.128	0.125	0.154	0.155	0.211	0.240	0.472
restaurant	0.639	0.271	0.125	0.106	0.132	0.134	0.133	0.157	0.174	0.208	0.247	0.486
.	0.437	0.170	0.110	0.095	0.126	0.119	0.116	0.146	0.150	0.213	0.244	0.446
Amy	0.695	0.266	0.119	0.100	0.123	0.122	0.118	0.147	0.149	0.205	0.252	0.438
gave	0.408	0.260	0.134	0.107	0.137	0.140	0.142	0.167	0.187	0.227	0.268	0.499

Table 6: Self-influence scores across layers 0–11 using three different methods. The best influence score within each layer is highlighted in bold.

Token	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
EAP												
Then	0.797	0.533	0.058	0.102	1.571	0.130	0.171	0.168	0.174	0.214	0.337	0.463
,	0.615	0.501	0.055	0.098	1.507	0.123	0.161	0.160	0.162	0.200	0.316	0.449
Danielle	0.163	0.047	0.053	0.094	1.456	0.119	0.151	0.151	0.151	0.178	0.303	0.412
and	0.625	0.501	0.055	0.099	1.517	0.126	0.165	0.162	0.168	0.204	0.320	0.456
Andrew	0.618	0.480	0.053	0.097	1456	0.117	0.144	0.147	0.145	0.179	0.305	0.420
had	0.668	0.547	0.060	0.106	1.614	0.136	0.181	0.172	0.190	0.222	0.343	0.493
a	0.648	0.516	0.057	0.102	1.559	0.136	0.168	0.164	0.173	0.207	0.330	0.471
lot	0.803	0.533	0.059	0.104	1.594	0.134	0.176	0.171	0.184	0.217	0.341	0.484
of	0.667	0.510	0.056	0.099	1.522	0.127	0.159	0.161	0.161	0.201	0.323	0.457
fun	0.646	0.508	0.056	0.101	1.545	0.131	0.167	0.166	0.175	0.206	0.331	0.463
at	0.630	0.516	0.056	0.098	1.534	0.126	0.162	0.164	0.163	0.206	0.326	0.457
the	0.590	0.499	0.055	0.098	1.505	0.129	0.158	0.157	0.161	0.197	0.316	0.451
office	0.626	0.514	0.055	0.100	1.522	0.129	0.166	0.161	0.173	0.201	0.315	0.455
.	0.574	0.472	0.052	0.092	1.424	0.116	0.142	0.152	0.136	0.181	0.306	0.421
Andrew	0.618	0.489	0.053	0.097	1.456	0.117	0.144	0.147	0.145	0.179	0.305	0.420
gave	0.640	0.550	0.061	0.106	1.623	0.135	0.179	0.176	0.182	0.224	0.356	0.487
a	0.648	0.516	0.057	0.102	1.559	0.136	0.168	0.164	0.173	0.207	0.330	0.471
computer	0.711	0.519	0.058	0.106	1.586	0.136	0.174	0.168	0.179	0.210	0.337	0.482
to	0.909	0.570	0.064	0.113	1.714	0.143	0.192	0.186	0.189	0.235	0.340	0.460
EAP-IG												
Then	0.618	0.282	2.637	0.140	0.137	0.130	0.171	0.168	0.199	0.214	0.371	0.463
,	0.413	0.177	2.505	0.137	0.131	0.123	0.161	0.160	0.189	0.200	0.351	0.449
Danielle	1.090	0.327	2.399	0.125	0.123	0.119	0.151	0.151	0.180	0.178	0.339	0.412
and	0.415	0.172	2.510	0.138	0.134	0.126	0.165	0.162	0.193	0.204	0.356	0.456
Andrew	0.430	0.217	2.412	0.135	0.124	0.117	0.144	0.147	0.176	0.179	0.340	0.420
had	0.481	0.219	2.702	0.150	0.143	0.136	0.181	0.172	0.211	0.222	0.383	0.493
a	0.472	0.177	2.594	0.140	0.139	0.136	0.168	0.164	0.201	0.207	0.366	0.471
lot	0.478	0.240	2.655	0.143	0.141	0.134	0.176	0.171	0.207	0.217	0.376	0.484
of	0.438	0.169	2.526	0.144	0.129	0.127	0.159	0.161	0.190	0.201	0.354	0.457
fun	0.502	0.229	2.548	0.139	0.135	0.131	0.167	0.166	0.198	0.206	0.364	0.463
at	0.437	0.209	2.561	0.140	0.130	0.126	0.162	0.164	0.194	0.206	0.359	0.457
the	0.395	0.160	2.486	0.137	0.133	0.129	0.158	0.157	0.191	0.197	0.349	0.451
office	0.527	0.193	2.517	0.138	0.134	0.129	0.166	0.161	0.194	0.201	0.351	0.455
.	0.383	0.172	2.380	0.126	0.123	0.116	0.142	0.152	0.175	0.181	0.339	0.421
Andrew	0.430	0.217	2.412	0.135	0.124	0.117	0.144	0.147	0.176	0.179	0.340	0.420
gave	0.386	0.258	2.752	0.149	0.135	0.135	0.179	0.176	0.213	0.224	0.391	0.487
a	0.472	0.177	2.594	0.140	0.139	0.136	0.168	0.164	0.201	0.207	0.366	0.471
computer	0.545	0.214	2.635	0.149	0.139	0.136	0.174	0.168	0.202	0.210	0.373	0.482
to	0.657	0.223	2.860	0.151	0.152	0.143	0.192	0.186	0.217	0.235	0.379	0.460
EAP-IG-KL												
Then	0.617	0.290	2.637	0.140	0.137	0.130	0.171	0.168	0.199	0.222	0.219	0.560
,	0.413	0.189	2.505	0.137	0.131	0.123	0.161	0.160	0.189	0.210	0.208	0.526
Danielle	1.090	0.339	2.399	0.125	0.123	0.119	0.151	0.151	0.180	0.199	0.213	0.509
and	0.415	0.191	2.510	0.138	0.134	0.126	0.165	0.162	0.193	0.209	0.209	0.531
Andrew	0.430	0.240	2.412	0.135	0.124	0.117	0.144	0.147	0.176	0.213	0.213	0.529
had	0.481	0.259	2.702	0.150	0.143	0.136	0.181	0.172	0.211	0.217	0.220	0.558
a	0.472	0.212	2.594	0.140	0.139	0.136	0.168	0.164	0.201	0.209	0.210	0.539
lot	0.478	0.269	2.655	0.143	0.141	0.134	0.176	0.171	0.207	0.217	0.215	0.540
of	0.438	0.186	2.526	0.144	0.129	0.127	0.159	0.161	0.190	0.211	0.208	0.537
fun	0.502	0.258	2.548	0.139	0.135	0.131	0.167	0.166	0.198	0.205	0.207	0.529
at	0.437	0.220	2.561	0.140	0.130	0.126	0.162	0.164	0.194	0.217	0.214	0.537
the	0.395	0.189	2.486	0.137	0.133	0.129	0.158	0.157	0.191	0.208	0.204	0.526
office	0.527	0.233	2.517	0.138	0.134	0.129	0.166	0.161	0.194	0.201	0.201	0.509
.	0.383	0.173	2.380	0.126	0.123	0.116	0.142	0.152	0.175	0.207	0.205	0.514
Andrew	0.430	0.240	2.412	0.135	0.124	0.117	0.144	0.147	0.176	0.213	0.213	0.529
gave	0.386	0.280	2.752	0.149	0.135	0.135	0.179	0.176	0.213	0.227	0.230	0.577
a	0.472	0.212	2.594	0.140	0.139	0.136	0.168	0.164	0.201	0.209	0.210	0.539
computer	0.545	0.255	2.635	0.149	0.139	0.136	0.174	0.168	0.202	0.216	0.214	0.546
to	0.657	0.211	2.860	0.151	0.152	0.143	0.192	0.186	0.217	0.227	0.229	0.533

Table 7: Self-influence scores across layers 0–11 using EAP method. The best influence score within each layer is highlighted.

Token	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11
EAP												
When	0.533	0.514	0.056	0.093	1.478	0.116	0.172	0.153	0.173	0.215	0.346	0.461
Anthony	1.255	0.441	0.049	0.083	1.331	0.104	0.149	0.132	0.148	0.181	0.310	0.395
and	0.390	0.468	0.053	0.089	1.430	0.112	0.167	0.145	0.171	0.207	0.322	0.448
Jose	0.841	0.439	0.050	0.084	1.353	0.107	0.156	0.138	0.154	0.188	0.315	0.407
got	0.533	0.513	0.057	0.096	1.500	0.120	0.179	0.158	0.184	0.222	0.342	0.473
a	0.601	0.490	0.055	0.093	1.480	0.119	0.173	0.147	0.177	0.210	0.335	0.470
drink	0.537	0.485	0.055	0.095	1.490	0.120	0.176	0.155	0.182	0.215	0.332	0.470
at	0.489	0.479	0.054	0.088	1.434	0.110	0.159	0.144	0.166	0.206	0.327	0.445
the	0.486	0.469	0.052	0.088	1.417	0.111	0.163	0.140	0.164	0.199	0.319	0.448
restaurant	0.712	0.481	0.055	0.095	1.485	0.121	0.176	0.153	0.180	0.212	0.326	0.459
,	0.423	0.465	0.052	0.088	1.408	0.110	0.161	0.142	0.165	0.203	0.319	0.436
Jose	0.841	0.439	0.050	0.084	1.353	0.107	0.156	0.138	0.154	0.188	0.315	0.407
decided	0.816	0.534	0.061	0.101	1.601	0.129	0.187	0.167	0.197	0.236	0.365	0.499
to	0.658	0.533	0.060	0.096	1.605	0.130	0.182	0.158	0.194	0.221	0.352	0.481
give	0.421	0.523	0.058	0.097	1.528	0.120	0.180	0.159	0.188	0.228	0.355	0.484
it	0.653	0.516	0.057	0.101	1.544	0.131	0.188	0.161	0.196	0.227	0.342	0.484
to	0.658	0.533	0.060	0.096	1.605	0.130	0.182	0.158	0.194	0.221	0.352	0.481
EAP-IG												
When	0.391	0.278	2.545	0.140	0.111	0.116	0.172	0.153	0.186	0.215	0.377	0.461
Anthony	0.878	0.269	2.223	0.128	0.099	0.104	0.149	0.132	0.154	0.181	0.331	0.395
and	0.321	0.175	2.388	0.129	0.108	0.112	0.167	0.145	0.175	0.207	0.355	0.448
Jose	0.643	0.264	2.256	0.127	0.102	0.107	0.156	0.138	0.159	0.188	0.341	0.407
got	0.450	0.243	2.559	0.140	0.114	0.120	0.179	0.158	0.190	0.222	0.377	0.473
a	0.440	0.191	2.488	0.136	0.114	0.119	0.173	0.147	0.183	0.210	0.369	0.470
drink	0.554	0.231	2.491	0.136	0.114	0.120	0.176	0.155	0.187	0.215	0.365	0.470
at	0.397	0.222	2.425	0.131	0.106	0.110	0.159	0.144	0.176	0.206	0.358	0.445
the	0.365	0.171	2.370	0.132	0.106	0.111	0.163	0.140	0.170	0.199	0.351	0.448
restaurant	0.569	0.233	2.469	0.136	0.112	0.121	0.176	0.153	0.183	0.212	0.361	0.459
,	0.336	0.180	2.364	0.129	0.105	0.110	0.161	0.142	0.173	0.203	0.348	0.436
Jose	0.643	0.264	2.256	0.127	0.102	0.107	0.156	0.138	0.159	0.188	0.341	0.407
decided	0.720	0.270	2.722	0.145	0.122	0.129	0.187	0.167	0.205	0.236	0.399	0.499
to	0.566	0.208	2.681	0.134	0.125	0.130	0.182	0.158	0.195	0.221	0.376	0.481
give	0.320	0.259	2.622	0.142	0.115	0.120	0.180	0.159	0.196	0.228	0.387	0.484
it	0.562	0.209	2.580	0.138	0.123	0.131	0.188	0.161	0.197	0.227	0.375	0.484
to	0.566	0.208	2.681	0.134	0.125	0.130	0.182	0.158	0.195	0.221	0.376	0.481
EAP-IG-KL												
When	0.391	0.256	2.545	0.140	0.111	0.116	0.172	0.153	0.186	0.192	0.258	0.581
Anthony	0.878	0.256	2.223	0.128	0.099	0.104	0.149	0.132	0.154	0.173	0.229	0.510
and	0.321	0.180	2.388	0.129	0.108	0.112	0.167	0.145	0.175	0.173	0.236	0.530
Jose	0.643	0.251	2.256	0.127	0.102	0.107	0.156	0.138	0.159	0.172	0.234	0.514
got	0.450	0.247	2.559	0.140	0.114	0.120	0.179	0.158	0.190	0.185	0.248	0.561
a	0.440	0.224	2.488	0.136	0.114	0.119	0.173	0.147	0.183	0.172	0.247	0.542
drink	0.554	0.248	2.491	0.136	0.114	0.120	0.176	0.155	0.187	0.176	0.235	0.536
at	0.397	0.212	2.425	0.131	0.106	0.110	0.159	0.144	0.176	0.175	0.242	0.544
the	0.365	0.192	2.370	0.132	0.106	0.111	0.163	0.140	0.170	0.169	0.236	0.526
restaurant	0.569	0.258	2.469	0.136	0.112	0.121	0.176	0.153	0.183	0.174	0.234	0.527
,	0.336	0.177	2.364	0.129	0.105	0.110	0.161	0.142	0.173	0.174	0.236	0.526
Jose	0.643	0.251	2.256	0.127	0.102	0.107	0.156	0.138	0.159	0.172	0.234	0.514
decided	0.720	0.275	2.722	0.145	0.122	0.129	0.187	0.167	0.205	0.196	0.260	0.581
to	0.566	0.215	2.681	0.134	0.125	0.130	0.182	0.158	0.195	0.185	0.258	0.535
give	0.320	0.253	2.622	0.142	0.115	0.120	0.180	0.159	0.196	0.192	0.254	0.579
it	0.562	0.240	2.580	0.138	0.123	0.131	0.188	0.161	0.197	0.182	0.241	0.544
to	0.566	0.215	2.681	0.134	0.125	0.130	0.182	0.158	0.195	0.185	0.258	0.535