Understanding Hidden Computations in Chain-of-Thought Reasoning

Aryasomayajula Ram Bharadwaj
Independent Researcher
ram.bharadwaj.arya@gmail.com

Abstract

Chain-of-Thought (CoT) prompting has significantly enhanced the reasoning abilities of large language models. However, recent studies have shown that models can still perform complex reasoning tasks even when the CoT is replaced with filler characters (e.g., "...”), leaving open questions about how models internally process and represent reasoning steps. In this paper, we investigate methods to decode these hidden computations in transformer models trained with filler CoT sequences. By analyzing layer-wise representations using the logit lens method and examining token rankings, we demonstrate that the hidden computations can be recovered without loss of performance. Our findings provide insights into the internal mechanisms of transformer models and open avenues for improving interpretability and transparency in language model reasoning.

1 Introduction

Chain-of-Thought (CoT) prompting has emerged as a powerful technique for improving the performance of large language models (LLMs) on complex reasoning tasks [1]. By encouraging models to generate intermediate reasoning steps, CoT prompting enables models to tackle problems that require multi-step reasoning. However, recent work by [2] demonstrates that these improvements can be achieved even when the CoT is replaced with hidden or filler characters (e.g., "...”). This raises intriguing questions about the nature of the computations being performed within these models and how they internally represent reasoning steps when the explicit chain of thought is obscured.

In this paper, we build upon the findings of [2] and investigate methods to decode these hidden computations in transformer models trained with filler CoT sequences. Focusing on the 3SUM task as a case study, we analyze the internal representations of the model using the logit lens method [3] and examine token rankings during the decoding process. Our goal is to understand how the model processes and retains information when the explicit reasoning steps are replaced with filler tokens, and whether the hidden computations can be recovered without loss of performance.

Our contributions are as follows:

- We provide a detailed analysis of the layer-wise representations in a transformer model trained with filler CoT sequences, revealing how the hidden computations evolve across layers.
- We demonstrate that the hidden computations can be recovered by examining higher-ranked tokens during decoding, without compromising the model's performance on the task.
- We discuss the implications of our findings for model interpretability and suggest directions for future research in understanding hidden computations in language models.

2 Background

In this section, we provide an overview of the key concepts and previous work relevant to our study, including Chain-of-Thought prompting, hidden chain-of-thought with filler tokens, the 3SUM task, and the logit lens method.

2.1 Chain-of-Thought Prompting

Chain-of-Thought (CoT) prompting involves providing language models with prompts that include intermediate reasoning steps leading to the final answer [1]. This technique has been shown to improve the performance of large language models on tasks requiring multi-step reasoning, such as mathematical problem-solving and commonsense reasoning.

Formally, given a question Q, the model is prompted to generate a sequence of tokens that includes a chain of reasoning R followed by the final answer A. The desired output is thus $S = R \circ A$, where $\circ$ denotes concatenation. For example, for the arithmetic question "What is the result of (12 plus 15) multiplied by 2 minus 5?", the chain of thought could be "First, add 12 and 15 to get 27. Next, multiply 27 by 2 to obtain 54. Then, subtract 5 from 54 to arrive at 49", leading to the final answer "49".

2.2 Hidden Chain-of-Thought with Filler Tokens

In a variant of CoT prompting, the intermediate reasoning steps are replaced with filler characters (e.g., "..."), resulting in a hidden chain of thought. That is, instead of outputting $S = R \circ A$, the model outputs $S = F \circ A$, where F is a sequence of filler tokens. [2] found that models trained with such filler CoT sequences can still perform well on reasoning tasks, suggesting that meaningful computation occurs internally despite the absence of explicit reasoning steps in the output.

This phenomenon raises questions about how models internally process and represent reasoning steps when the chain of thought is obscured. Understanding this internal processing is important for model interpretability and could have implications for model safety and controllability.

2.3 The 3SUM Task

The 3SUM task involves determining whether any three numbers in a given list sum to zero. Formally, given a list of integers $S = \{s_1, s_2, \ldots, s_n\}$, the task is to decide whether there exist indices i, j, k (with $i \neq j \neq k$) such that $s_i + s_j + s_k = 0$.

The 3SUM problem is a well-known computational problem and serves as a proxy for more complex reasoning tasks in the context of language models [2]. By using a mathematical problem that requires combinatorial reasoning, we can study the model's ability to perform internal computations and understand how it processes the problem when the chain of thought is hidden.

2.4 Logit Lens Method

The logit lens method, introduced by [3], provides a way to inspect the internal representations of a language model by mapping the activations at each layer back to the vocabulary space. Specifically, at each layer l, the hidden state $\mathbf{h}^l$ is projected onto the vocabulary logits using the output embedding matrix $\mathbf{W}_{\text{out}}$, yielding $\mathbf{z}^l = \mathbf{h}^l \mathbf{W}_{\text{out}}$.

By applying the softmax function, we obtain a probability distribution over the vocabulary at each layer. This method allows us to observe the model's intermediate predictions and gain insights into how information is processed and transformed across layers. By examining the top predicted tokens at each layer, we can infer the model's evolving "thoughts" as it processes the input and generates the output.

3 Related Work

Understanding the internal mechanisms of Chain-of-Thought (CoT) reasoning has been a significant focus of recent research. [2] demonstrated that models can perform reasoning tasks even when explicit CoT is replaced with filler tokens, suggesting the presence of hidden computations. This finding challenges the assumption that visible CoT directly reflects the model's reasoning process and indicates that models may retain and utilize internal reasoning pathways that are not immediately observable in the output.

Further studies have explored the faithfulness and reliability of CoT reasoning. Lanham et al. [7] investigated the faithfulness of CoT explanations, revealing that as models become larger and more capable, the faithfulness of their reasoning often decreases across various tasks. Their work highlights that CoT's performance improvements do not solely stem from added computational steps or specific phrasing, but also raise concerns about the genuine alignment between stated reasoning and actual cognitive processes.

Turpin et al. [9] found that CoT explanations can systematically misrepresent the true reasons behind a model's predictions. They demonstrated that models might generate plausible yet misleading explanations, especially when influenced by biased inputs, leading to significant drops in accuracy and reinforcing stereotypes without acknowledging underlying biases.

Addressing these challenges, Radhakrishnan et al. [8] proposed decomposing questions into simpler subquestions to enhance the faithfulness of model-generated reasoning. Their decomposition-based methods not only improve the reliability of CoT explanations but also maintain performance gains, suggesting a viable pathway to more transparent and verifiable reasoning processes in large language models.

Collectively, these studies underscore the complexity of CoT reasoning and the necessity for ongoing efforts to ensure that model-generated explanations are both faithful and transparent.

4 Methodology

In this section, we describe the experimental setup, including the model architecture, training procedure, and the analysis methods used to investigate the hidden computations.

4.1 Model and Training

We employed a transformer language model based on the LLaMA architecture [6], with 4 layers, a hidden dimension of 384, and 6 attention heads, totaling 34 million parameters. The model was randomly initialized and trained from scratch.

5 Dataset Generation

We generated a synthetic dataset for the **Match-3** task, an extension of the classic 3SUM problem. Each input instance comprises a sequence of integer tuples, where each integer within a tuple is sampled uniformly from the range $[0, 9]$. Specifically, each input sequence $S = \{s_1, s_2, \ldots, s_7\}$ consists of 7 tuples, each of dimension 3. The corresponding labels indicate whether any triplet of tuples within the sequence sums to zero modulo 10, thereby ensuring a balanced distribution of positive and negative examples.

To create this dataset, we utilized the `Match3` class, which facilitates the generation of both true and corrupted instances:

- **True Instances:** These are sequences where at least one triplet of tuples satisfies the 3SUM condition modulo 10. The generation process ensures uniform sampling of tuples and incorporates symmetry to maintain balanced labels.

- **Corrupted Instances:** These sequences include intentional perturbations based on a specified corruption rate (set to $\frac{4}{3}$ in our experiments). Corruptions involve altering tuple values to disrupt existing valid triplets or to introduce new ones, depending on the corruption parameters. This approach helps in creating challenging examples that test the model's robustness.

Additionally, the dataset includes **Chain-of-Thought (CoT)** annotations to enhance model interpretability. We implemented various CoT strategies, such as `rand_cot` and `serial_cot`, which provide intermediate reasoning steps alongside the final labels. These annotations are configurable, allowing for different levels of detail and types of explanations within the dataset.

5.1 Instance-Adaptive Chain of Thought

Instance-adaptive Chain-of-Thought (CoT) differs from parallelizable CoT in that it requires caching subproblem solutions within token outputs. Specifically, in instance-adaptive computation, the operations performed in later CoT tokens depend on the results obtained from earlier CoT tokens. This dependency structure is incompatible with the parallel processing nature of filler token computation.

In the context of the 3SUM problem, our instance-adaptive CoT approach decomposes the problem into dimension-wise 3SUMs. For each triplet of tuples, a dimension-wise summation is computed only if the previous dimension's summation equals zero. This creates an instance-adaptive dependency where the computation of one dimension influences the subsequent computations. For example, if the sum of the first dimension is zero, the model proceeds to compute the sum of the second dimension; otherwise, it skips to the next triplet. This sequential dependency ensures that the CoT reflects the logical progression of solving the problem step-by-step.

5.2 Data Generation Procedure

The data generation process for both parallelizable and instance-adaptive CoT follows identical input sampling procedures. The Chain-of-Thought generation is implemented as follows:

Given an input sequence, for example, "A15 B75 C22 D13", the corresponding CoT is generated as ": A B C 15 75 22 2 B C D 75 22 13 0 ANS True". The generation process for each triplet, such as "A B C" and "B C D", involves the following steps:

1. **Triple Listing:** For each triplet, if the sum of the first dimension is zero, list the individual triple (e.g., ": A B C").
2. **Value Listing:** List the values of the triples by copying from the input sequence (e.g., "15 75 22").
3. **Summation Result:** Compute and list the result of summing the given triple in each dimension modulo 10 (e.g., "2" since $(15 + 75 + 22) \mod 10 = 2$ for the second dimension).

In our example, the triplet "A B C" sums to 0 in the first dimension but sums to 2 in the second dimension. Conversely, the triplet "B C D" sums to 0 in both dimensions, thereby satisfying the 3SUM condition for this input. The CoT annotations provide intermediate reasoning steps that trace the computation of these sums, facilitating better interpretability and debugging of the model's decision-making process.

The generation process also includes options for adding fillers and controlling the inclusion of CoT annotations, ensuring a diverse and comprehensive dataset. Specifically, the dataset includes mixtures of filler-token sequences and instance-adaptive sequences.

The dataset generation parameters were set as follows:

- **Training Samples:** 10,000,000 instances
- **Testing Samples:** 2,000 instances
- **Tuple Dimension:** 3
- **Modulus (mod):** 10
- **Sequence Length:** 7
- **True Instance Rate:** 50%
- **CoT Rate:** 50%
- **No Filler Rate:** 0%
- **Corruption Rate:** $\frac{4}{3}$

Inputs are uniformly sampled within the specified range, and the generation process incorporates both true and corrupted instances to balance the dataset. Corrupted instances are generated by introducing perturbations based on the corruption rate, which alters tuple values to either disrupt existing valid triplets or create new ones. This strategy enhances the dataset's diversity and challenges the model to generalize effectively.

Chain-of-Thought annotations are generated using two primary strategies: `rand_cot` and `serial_cot`. The `rand_cot` strategy randomly includes positions of the 2SUM summands in the CoT, whereas the `serial_cot` strategy enforces a sequential dependency, ensuring that each step in the CoT builds upon the previous one. These strategies are configurable, allowing for various levels of detail and types of explanations within the dataset.

The final dataset is split into training and testing subsets, each saved in CSV format for ease of use in downstream tasks such as model training and evaluation. Additionally, a JSON file documenting the hyperparameters used during generation is included to ensure reproducibility and facilitate further experimentation.

Target Sequences: For each input sequence, the target output was a sequence of filler tokens (e.g., "...") of a fixed length, followed by the final answer ("True" or "False"). The number of filler tokens corresponded to the expected length of a chain of thought, ensuring that the model processes a sequence similar in length to one with explicit reasoning steps.

Training Procedure: The model was trained to predict the target sequences using the cross-entropy loss function. We optimized the model using the Adam optimizer with a learning rate of 1×10^{-4} and a batch size of 256. The training was conducted over 5 epochs.

5.3 Layer-wise Representation Analysis

To investigate how the model processes the hidden computations across layers, we employed the logit lens method [3]. At each layer l, we extracted the hidden states $\mathbf{h}^l$ and projected them onto the vocabulary logits using the output embedding matrix $\mathbf{W}_{\text{out}}$, yielding $\mathbf{z}^l = \mathbf{h}^l \mathbf{W}_{\text{out}}$.

By applying the softmax function to $\mathbf{z}^l$, we obtained the probability distribution over the vocabulary at each layer. This allowed us to examine the top predicted tokens at each layer and observe how the model's internal representations evolve across layers. Specifically, we were interested in whether the filler tokens or the original reasoning steps were predominant in the predictions at different layers.

5.4 Token Ranking Analysis

Building upon the observations from the layer-wise analysis, we conducted a token ranking analysis during the decoding process. For each position in the output sequence, we examined the top k candidate tokens based on their predicted probabilities. Our aim was to determine whether the original, non-filler CoT tokens appeared among the lower-ranked candidates when the filler token was the top prediction.

By analyzing the token rankings, we assessed whether the model retains the hidden computations beneath the filler tokens and whether these computations can be recovered by considering lower-ranked tokens. This analysis provides insights into the model's internal processing and the extent to which the hidden computations are accessible.

5.5 Modified Decoding Algorithm

Based on the insights from the token ranking analysis, we implemented a modified greedy autoregressive decoding algorithm to recover the hidden computations. The algorithm operates as follows:

1. Initialize the output sequence with the start token.
2. For each decoding step t:
   (a) Compute the probability distribution over the vocabulary using the current hidden state.
   (b) If the top-ranked token is the filler token, select the highest-ranked non-filler token as the output for position t.
   (c) Otherwise, select the top-ranked token as usual.
   (d) Update the hidden state based on the selected token.
3. Continue the process until the end-of-sequence token is generated or the maximum sequence length is reached.

This modified decoding algorithm allows us to recover the hidden computations by bypassing the filler tokens when they are the top prediction. By selecting the next most probable non-filler token, we can reconstruct the original reasoning steps without compromising the model's performance on the task.

6 Results and Discussion

In this section, we present the results of our experiments and discuss their implications for understanding hidden computations in transformer models.

6.1 Layer-wise Representation Analysis

Our layer-wise analysis revealed a gradual evolution of representations across the model's layers. In the initial layers (layers 1 and 2), the model's activations corresponded to the raw numerical sequences associated with the 3SUM problem's chain of thought. The top predicted tokens at these layers were primarily numerical tokens representing elements of the input sequence and intermediate calculations.

Starting from layer 3, we observed the emergence of filler tokens among the top-ranked predictions. The filler token ("...") began to appear more frequently as the top prediction, indicating that the model was starting to shift its focus toward producing the expected output format with filler tokens.

By layer 4 (the final layer), the filler token dominated the top predictions, and the original numerical tokens were relegated to lower ranks (rank-2). This pattern suggests that the model performs the necessary computations in the earlier layers and then overwrites the intermediate representations with filler tokens in the later layers to produce the expected output.

Figure 1 illustrates the percentage of filler tokens among the top predictions at each layer. The transition from numerical tokens to filler tokens across layers highlights how the model balances internal computation with output formatting.

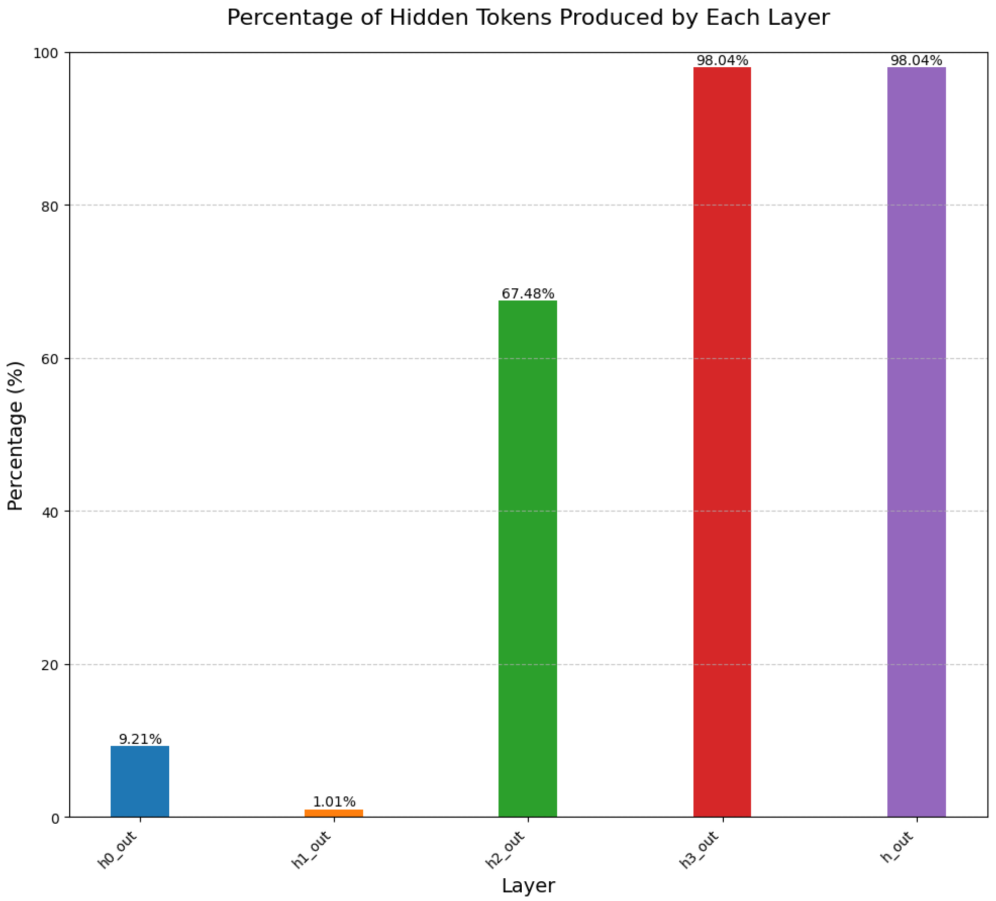


Figure 1: Percentage of filler tokens among top predictions across layers

6.2 Token Ranking Analysis

In the token ranking analysis, we found that while the filler token was consistently the top-ranked token at each decoding step in the later layers, the original, non-filler CoT tokens remained among the lower-ranked candidates. Specifically, the second-ranked token often corresponded to the numerical tokens representing the hidden reasoning steps.

This finding supports the hypothesis that the model retains the hidden computations beneath the filler tokens. The presence of the original reasoning tokens among the lower-ranked predictions indicates that the model internally processes the reasoning steps but prioritizes the filler tokens in the output to match the training targets.

6.3 Modified Decoding Algorithm

By applying the modified decoding algorithm described in Section 5.5, we were able to recover the hidden computations without compromising the model's performance on the 3SUM task. The model's final answers remained correct, and the reconstructed reasoning steps provided a transparent view of the model's internal computations.

We compared our modified decoding method with two baselines:

- **Standard Greedy Decoding:** Outputs the filler tokens as in the original training data.
- **Random Token Replacement:** Replaces filler tokens with randomly selected tokens from the vocabulary.

As shown in Figure 2, our method significantly outperformed the random replacement and provided meaningful reasoning steps aligned with the model's internal computations.

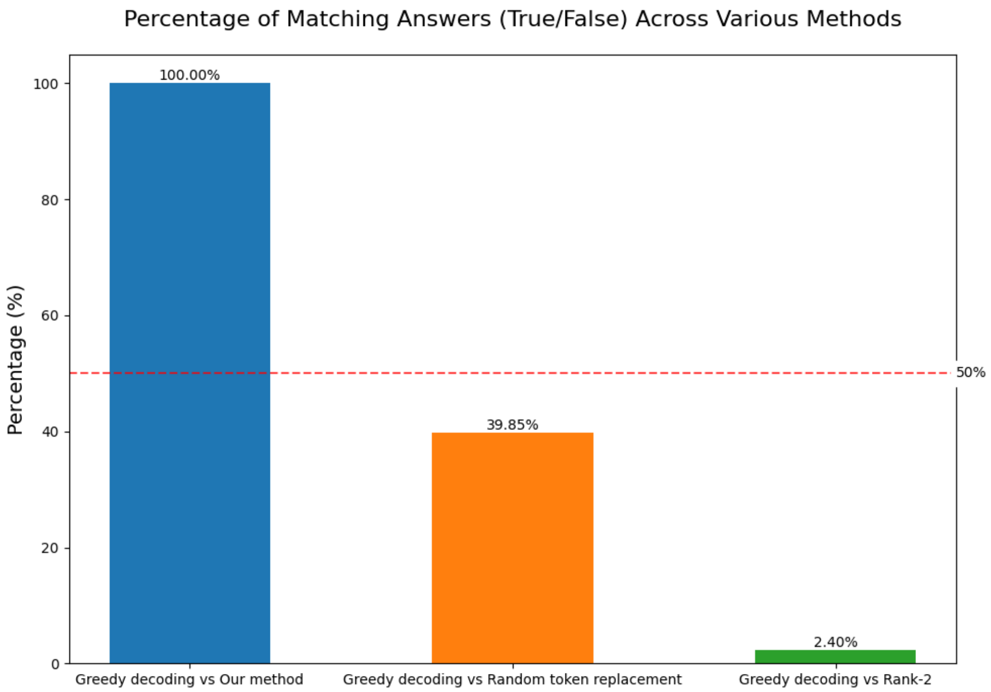


Figure 2: Comparison of decoding methods: Our method achieves higher accuracy in recovering hidden computations compared to random token replacement

6.4 Discussion

Our findings indicate that the model internally performs the necessary computations for the 3SUM task and that these computations can be recovered by examining lower-ranked tokens during decoding. The model appears to perform the reasoning in the earlier layers and then overwrites the intermediate representations with filler tokens in the later layers to produce the expected output.

This overwriting behavior may involve mechanisms such as induction heads [4], where the model learns to copy or overwrite tokens based on patterns in the data. Understanding how the model manages the trade-off between performing computations and producing the expected output could provide valuable insights into the internal workings of transformer models.

Our analysis contributes to the broader understanding of model interpretability and highlights the importance of examining internal representations to gain insights into how models process and represent information. By uncovering the hidden computations, we can develop methods to better control and interpret the outputs of language models.

Limitations: One limitation of our study is that it focuses on a synthetic task (3SUM) and a relatively small transformer model. While this allows for controlled experimentation, it may not fully capture the complexities of real-world language tasks and larger models. Future work should explore whether similar phenomena occur in larger models and more complex tasks.

7 Conclusion

We have presented an analysis of hidden computations in transformer models trained with filler CoT sequences on the 3SUM task. By utilizing the logit lens method and examining token rankings, we demonstrated that the hidden computations can be recovered without loss of performance. Our findings shed light on how models internally process and represent reasoning steps when the explicit chain of thought is obscured.

This work contributes to the broader understanding of model interpretability and opens avenues for improving transparency in language model reasoning. By uncovering the hidden computations, we can gain insights into the internal mechanisms of chain of thought reasining in language models.

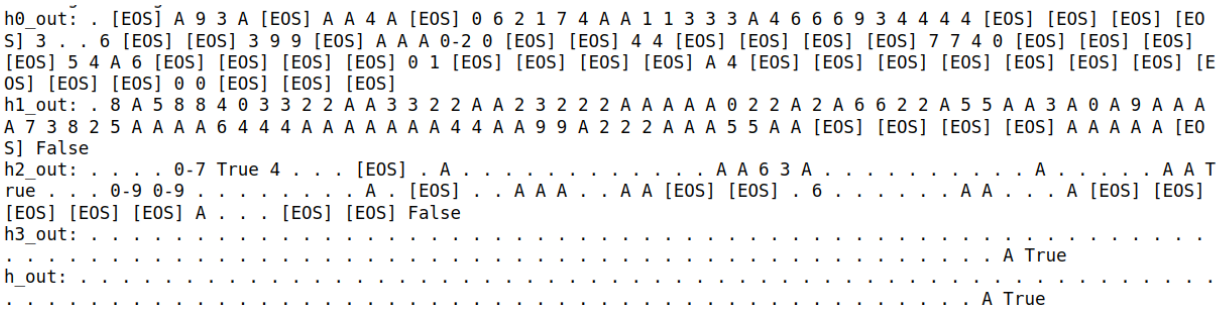
8 Future Work

Future research should focus on exploring the mechanisms responsible for overwriting the hidden computations with filler tokens. Investigating whether specific circuits, such as induction heads or particular attention patterns, are involved could enhance our understanding of how models balance computation and output formatting.

Moreover, applying similar analysis techniques to other tasks (more general tasks other than 3SUM) and models could assess the generality of our findings. Extending the investigation to natural language tasks and larger models would help determine whether the observed phenomena persist in more complex settings.

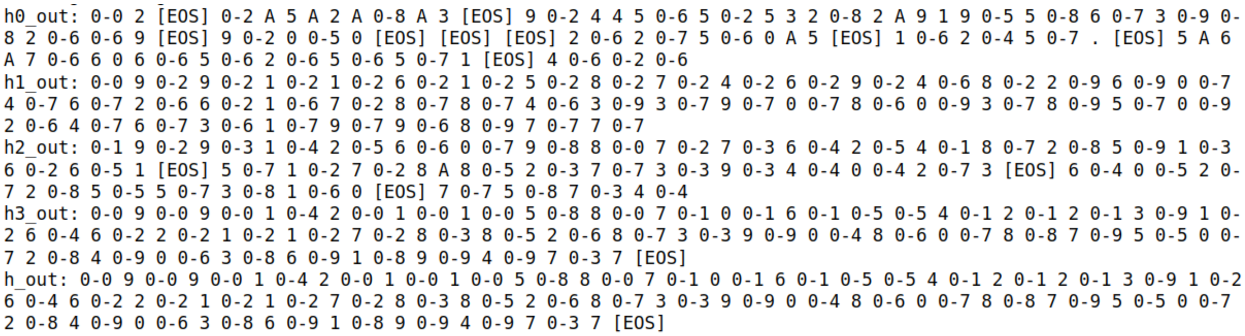
A Layer-wise View of Sequences Generated via Various Decoding Methods

In this appendix, we provide visualizations of the sequences generated using different decoding methods to illustrate how the hidden computations can be recovered.



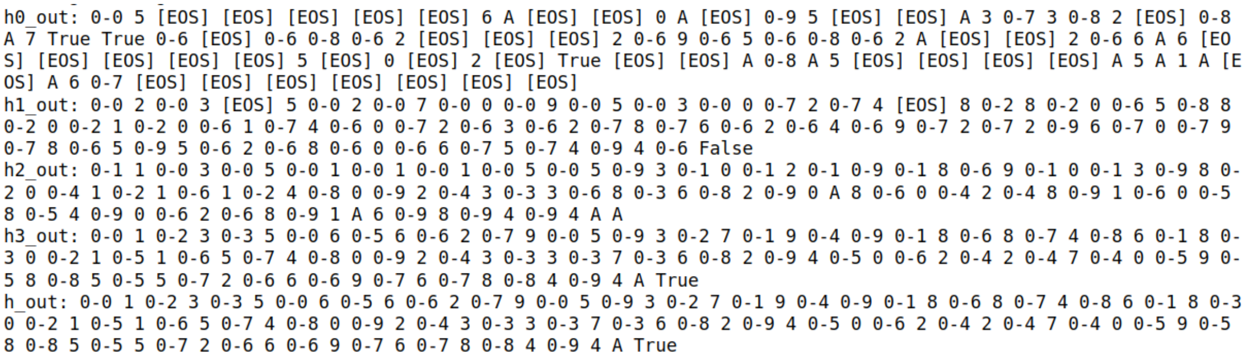
```
h0_out: . [EOS] A 9 3 A [EOS] A A 4 A [EOS] 0 6 2 1 7 4 A A 1 1 3 3 3 A 4 6 6 6 9 3 4 4 4 4 [EOS] [EOS] [EOS] [EO
S] 3 . . 6 [EOS] [EOS] 3 9 9 [EOS] A A A 0-2 0 [EOS] [EOS] 4 4 [EOS] [EOS] [EOS] [EOS] 7 7 4 0 [EOS] [EOS] [EOS]
[EOS] 5 4 A 6 [EOS] [EOS] [EOS] [EOS] 0 1 [EOS] [EOS] [EOS] [EOS] A 4 [EOS] [EOS] [EOS] [EOS] [EOS] [EOS] [EOS] [E
OS] [EOS] [EOS] 0 0 [EOS] [EOS] [EOS]
h1_out: . 8 A 5 8 8 4 0 3 3 2 2 A A 3 3 2 2 A A 2 3 2 2 2 A A A A A 0 2 2 A 2 A 6 6 2 2 A 5 5 A A 3 A 0 A 9 A A A
A 7 3 8 2 5 A A A A 6 4 4 4 A A A A A A A 4 4 A A 9 9 A 2 2 2 A A A 5 5 A A [EOS] [EOS] [EOS] [EOS] A A A A A [EO
S] False
h2_out: . . . . 0-7 True 4 . . . [EOS] . A . . . . . . . . . . . . A A 6 3 A . . . . . . . . . . A . . . . . A A T
rue . . . 0-9 0-9 . . . . . . . . A . [EOS] . . A A A . . A A [EOS] [EOS] . 6 . . . . . . A A . . . A [EOS] [EOS]
[EOS] [EOS] [EOS] A . . . [EOS] [EOS] False
h3_out: . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . A True
h_out: . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . A True
```

Figure 3: Greedy Decoding: The model outputs filler tokens followed by the final answer



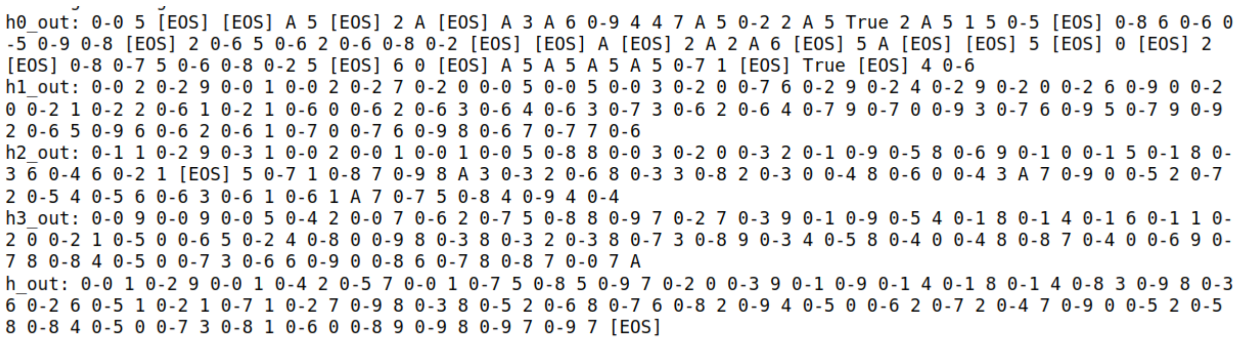
```
h0_out: 0-0 2 [EOS] 0-2 A 5 A 2 A 0-8 A 3 [EOS] 9 0-2 4 4 5 0-6 5 0-2 5 3 2 0-8 2 A 9 1 9 0-5 5 0-8 6 0-7 3 0-9 0-
8 2 0-6 0-6 9 [EOS] 9 0-2 0 0-5 0 [EOS] [EOS] [EOS] 2 0-6 2 0-7 5 0-6 0 A 5 [EOS] 1 0-6 2 0-4 5 0-7 . [EOS] 5 A 6
A 7 0-6 6 0 6 0-6 5 0-6 2 0-6 5 0-6 5 0-7 1 [EOS] 4 0-6 0-2 0-6
h1_out: 0-0 9 0-2 9 0-2 1 0-2 1 0-2 6 0-2 1 0-2 5 0-2 8 0-2 7 0-2 4 0-2 6 0-2 9 0-2 4 0-6 8 0-2 2 0-9 6 0-9 0 0-7
4 0-7 6 0-7 2 0-6 6 0-2 1 0-6 7 0-2 8 0-7 8 0-7 4 0-6 3 0-9 3 0-7 9 0-7 0 0-7 8 0-6 0 0-9 3 0-7 8 0-9 5 0-7 0 0-9
2 0-6 4 0-7 6 0-7 3 0-6 1 0-7 9 0-7 9 0-6 8 0-9 7 0-7 7 0-7
h2_out: 0-1 9 0-2 9 0-3 1 0-4 2 0-5 6 0-6 0 0-7 9 0-8 8 0-0 7 0-2 7 0-3 6 0-4 2 0-5 4 0-1 8 0-7 2 0-8 5 0-9 1 0-3
6 0-2 6 0-5 1 [EOS] 5 0-7 1 0-2 7 0-2 8 A 8 0-5 2 0-3 7 0-7 3 0-3 9 0-3 4 0-4 0 0-4 2 0-7 3 [EOS] 6 0-4 0 0-5 2 0-
7 2 0-8 5 0-5 5 0-7 3 0-8 1 0-6 0 [EOS] 7 0-7 5 0-8 7 0-3 4 0-4
h3_out: 0-0 9 0-0 9 0-0 1 0-4 2 0-0 1 0-0 1 0-0 5 0-8 8 0-0 7 0-1 0 0-1 6 0-1 0-5 0-5 4 0-1 2 0-1 2 0-1 3 0-9 1 0-
2 6 0-4 6 0-2 2 0-2 1 0-2 1 0-2 7 0-2 8 0-3 8 0-5 2 0-6 8 0-7 3 0-3 9 0-9 0 0-4 8 0-6 0 0-7 8 0-8 7 0-9 5 0-5 0 0-
7 2 0-8 4 0-9 0 0-6 3 0-8 6 0-9 1 0-8 9 0-9 4 0-9 7 0-3 7 [EOS]
h_out: 0-0 9 0-0 9 0-0 1 0-4 2 0-0 1 0-0 1 0-0 5 0-8 8 0-0 7 0-1 0 0-1 6 0-1 0-5 0-5 4 0-1 2 0-1 2 0-1 3 0-9 1 0-2
6 0-4 6 0-2 2 0-2 1 0-2 1 0-2 7 0-2 8 0-3 8 0-5 2 0-6 8 0-7 3 0-3 9 0-9 0 0-4 8 0-6 0 0-7 8 0-8 7 0-9 5 0-5 0 0-7
2 0-8 4 0-9 0 0-6 3 0-8 6 0-9 1 0-8 9 0-9 4 0-9 7 0-3 7 [EOS]
```

Figure 4: Greedy Decoding with Rank-2 Tokens



```
h0_out: 0-0 5 [EOS] [EOS] [EOS] [EOS] [EOS] 6 A [EOS] [EOS] 0 A [EOS] 0-9 5 [EOS] [EOS] A 3 0-7 3 0-8 2 [EOS] 0-8
A 7 True True 0-6 [EOS] 0-6 0-8 0-6 2 [EOS] [EOS] [EOS] 2 0-6 9 0-6 5 0-6 0-8 0-6 2 A [EOS] [EOS] 2 0-6 6 A 6 [EO
S] [EOS] [EOS] [EOS] [EOS] 5 [EOS] 0 [EOS] 2 [EOS] True [EOS] [EOS] A 0-8 A 5 [EOS] [EOS] [EOS] [EOS] A 5 A 1 A [E
OS] A 6 0-7 [EOS] [EOS] [EOS] [EOS] [EOS] [EOS] [EOS]
h1_out: 0-0 2 0-0 3 [EOS] 5 0-0 2 0-0 7 0-0 0 0-0 9 0-0 5 0-0 3 0-0 0 0-7 2 0-7 4 [EOS] 8 0-2 8 0-2 0 0-6 5 0-8 8
0-2 0 0-2 1 0-2 0 0-6 1 0-7 4 0-6 0 0-7 2 0-6 3 0-6 2 0-7 8 0-7 6 0-6 2 0-6 4 0-6 9 0-7 2 0-7 2 0-9 6 0-7 0 0-7 9
0-7 8 0-6 5 0-9 5 0-6 2 0-6 8 0-6 0 0-6 6 0-7 5 0-7 4 0-9 4 0-6 False
h2_out: 0-1 1 0-0 3 0-0 5 0-0 1 0-0 1 0-0 1 0-0 5 0-0 5 0-9 3 0-1 0 0-1 2 0-1 0-9 0-1 8 0-6 9 0-1 0 0-1 3 0-9 8 0-
2 0 0-4 1 0-2 1 0-6 1 0-2 4 0-8 0 0-9 2 0-4 3 0-3 3 0-6 8 0-3 6 0-8 2 0-9 0 A 8 0-6 0 0-4 2 0-4 8 0-9 1 0-6 0 0-5
8 0-5 4 0-9 0 0-6 2 0-6 8 0-9 1 A 6 0-9 8 0-9 4 0-9 4 A A
h3_out: 0-0 1 0-2 3 0-3 5 0-0 6 0-5 6 0-6 2 0-7 9 0-0 5 0-9 3 0-2 7 0-1 9 0-4 0-9 0-1 8 0-6 8 0-7 4 0-8 6 0-1 8 0-
3 0 0-2 1 0-5 1 0-6 5 0-7 4 0-8 0 0-9 2 0-4 3 0-3 3 0-3 7 0-3 6 0-8 2 0-9 4 0-5 0 0-6 2 0-4 2 0-4 7 0-4 0 0-5 9 0-
5 8 0-8 5 0-5 5 0-7 2 0-6 6 0-6 9 0-7 6 0-7 8 0-8 4 0-9 4 A True
h_out: 0-0 1 0-2 3 0-3 5 0-0 6 0-5 6 0-6 2 0-7 9 0-0 5 0-9 3 0-2 7 0-1 9 0-4 0-9 0-1 8 0-6 8 0-7 4 0-8 6 0-1 8 0-3
0 0-2 1 0-5 1 0-6 5 0-7 4 0-8 0 0-9 2 0-4 3 0-3 3 0-3 7 0-3 6 0-8 2 0-9 4 0-5 0 0-6 2 0-4 2 0-4 7 0-4 0 0-5 9 0-5
8 0-8 5 0-5 5 0-7 2 0-6 6 0-6 9 0-7 6 0-7 8 0-8 4 0-9 4 A True
```

Figure 5: Our Method: Greedy Decoding with Filler Tokens Replaced by Rank-2 Tokens (Recovering Hidden Computations)



```
h0_out: 0-0 5 [EOS] [EOS] A 5 [EOS] 2 A [EOS] A 3 A 6 0-9 4 4 7 A 5 0-2 2 A 5 True 2 A 5 1 5 0-5 [EOS] 0-8 6 0-6 0
-5 0-9 0-8 [EOS] 2 0-6 5 0-6 2 0-6 0-8 0-2 [EOS] [EOS] A [EOS] 2 A 2 A 6 [EOS] 5 A [EOS] [EOS] 5 [EOS] 0 [EOS] 2
[EOS] 0-8 0-7 5 0-6 0-8 0-2 5 [EOS] 6 0 [EOS] A 5 A 5 A 5 A 5 0-7 1 [EOS] True [EOS] 4 0-6
h1_out: 0-0 2 0-2 9 0-0 1 0-0 2 0-2 7 0-2 0 0-0 5 0-0 5 0-0 3 0-2 0 0-7 6 0-2 9 0-2 4 0-2 9 0-2 0 0-2 6 0-9 0 0-2
0 0-2 1 0-2 2 0-6 1 0-2 1 0-6 0 0-6 2 0-6 3 0-6 4 0-6 3 0-7 3 0-6 2 0-6 4 0-7 9 0-7 0 0-9 3 0-7 6 0-9 5 0-7 9 0-9
2 0-6 5 0-9 6 0-6 2 0-6 1 0-7 0 0-7 6 0-9 8 0-6 7 0-7 7 0-6
h2_out: 0-1 1 0-2 9 0-3 1 0-0 2 0-0 1 0-0 1 0-0 5 0-8 8 0-0 3 0-2 0 0-3 2 0-1 0-9 0-5 8 0-6 9 0-1 0 0-1 5 0-1 8 0-
3 6 0-4 6 0-2 1 [EOS] 5 0-7 1 0-8 7 0-9 8 A 3 0-3 2 0-6 8 0-3 3 0-8 2 0-3 0 0-4 8 0-6 0 0-4 3 A 7 0-9 0 0-5 2 0-7
2 0-5 4 0-5 6 0-6 3 0-6 1 0-6 1 A 7 0-7 5 0-8 4 0-9 4 0-4
h3_out: 0-0 9 0-0 9 0-0 5 0-4 2 0-0 7 0-6 2 0-7 5 0-8 8 0-9 7 0-2 7 0-3 9 0-1 0-9 0-5 4 0-1 8 0-1 4 0-1 6 0-1 1 0-
2 0 0-2 1 0-5 0 0-6 5 0-2 4 0-8 0 0-9 8 0-3 8 0-3 2 0-3 8 0-7 3 0-8 9 0-3 4 0-5 8 0-4 0 0-4 8 0-8 7 0-4 0 0-6 9 0-
7 8 0-8 4 0-5 0 0-7 3 0-6 6 0-9 0 0-8 6 0-7 8 0-8 7 0-0 7 A
h_out: 0-0 1 0-2 9 0-0 1 0-4 2 0-5 7 0-0 1 0-7 5 0-8 5 0-9 7 0-2 0 0-3 9 0-1 0-9 0-1 4 0-1 8 0-1 4 0-8 3 0-9 8 0-3
6 0-2 6 0-5 1 0-2 1 0-7 1 0-2 7 0-9 8 0-3 8 0-5 2 0-6 8 0-7 6 0-8 2 0-9 4 0-5 0 0-6 2 0-7 2 0-4 7 0-9 0 0-5 2 0-5
8 0-8 4 0-5 0 0-7 3 0-8 1 0-6 0 0-8 9 0-9 8 0-9 7 0-9 7 [EOS]
```

Figure 6: Random Token Replacement: Replacing Filler Tokens with Random Tokens

As shown in Figures 5 and 6, our method successfully recovers the hidden computations, while random token replacement leads to incoherent sequences.

References

[1] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[2] Jacob Pfau, William Merrill, and Samuel R. Bowman. Let's think dot by dot: Hidden computation in transformer language models. *arXiv preprint arXiv:2308.07317*, 2023.

[3] Nostalgebraist. Interpreting GPT: the logit lens. `https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens`, 2020.

[4] Nelson Elhage, Neel Nanda, Catherine Olsson, Nicholas Schiefer, Skyler Hallinan, Stanislav Fort, Danny Hernandez, and Chris Olah. A mathematical framework for transformer circuits. `https://transformer-circuits.pub/2021/framework/index.html`, 2021.

[5] Jonathan Turpin, Uma Roy, Gabriel Poesia, Jared Kaplan, and Noah Goodman. Language models are problem solvers: Formulation of prompts and the faithfulness of model reasoning. *arXiv preprint arXiv:2305.14325*, 2023. `https://arxiv.org/abs/2305.14325`.

[6] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Edouard Grave, Matthieu Lin, Pierre H. Bluck, Azhar Golriz, Morgan Funtowicz, Lucas Sbarra, Aurelien Rodriguez, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[7] Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamilė Lukošiūtė, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Larson, Sam McCandlish, Sandipan Kundu, Shannon Yang, Thomas Henighan, Timothy Maxwell, Timothy Telleen-Lawton, Tristan Hume, Zac Hatfield-Dodds, Jared Kaplan, Jan Brauner, Samuel R. Bowman, and Ethan Perez. Measuring Faithfulness in Chain-of-Thought Reasoning. *arXiv preprint arXiv:2307.13702*, 2023. `https://arxiv.org/abs/2307.13702`.

[8] Ansh Radhakrishnan, Karina Nguyen, Anna Chen, Carol Chen, Carson Denison, Danny Hernandez, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamilė Lukošiūtė, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Sam McCandlish, Sheer El Showk, Tamera Lanham, Tim Maxwell, Venkatesa Chandrasekaran, Zac Hatfield-Dodds, Jared Kaplan, Jan Brauner, Samuel R. Bowman, and Ethan Perez. Question Decomposition Improves the Faithfulness of Model-Generated Reasoning. *arXiv preprint arXiv:2307.11768*, 2023. `https://arxiv.org/abs/2307.11768`.

[9] Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting. *arXiv preprint arXiv:2305.04388*, 2023. `https://arxiv.org/abs/2305.04388`.