Auto-patching: Enhancing Multi-Hop Reasoning in Language Models

Dean Tahory, **Omer Talmi**, **Omar Abo Mokh**, and **Aviv Jan**

Tel Aviv University

{deantahory, omertalmi, omarabomokh, avivjan}@mail.tau.ac.il

Abstract

In recent years, large language models (LLMs) have demonstrated significant advances in natural language understanding and generation. However, these models often struggle with complex reasoning tasks, particularly those involving multi-hop questions that require linking multiple pieces of information. In this paper, we introduce the Auto Patch method, a novel approach that leverages dynamic hidden state patching to enhance the multi-hop reasoning capabilities of the LLaMA 2 model. Using the PatchScopes framework, Auto Patch dynamically adjusts the model's hidden states during inference to improve the model's ability to synthesize information across different steps. We evaluated our method on the MuSiQue dataset, focusing on 2-hop questions, and compared its performance against baseline and Chain-of-Thought (CoT) prompting methods. The results show that Auto Patch significantly outperforms the baseline, achieving an solve rate of 23.63%, compared to 18.45% for the baseline. Although it did not surpass the CoT method, which achieved 27.44%, Auto Patch offers a promising direction for further enhancing the reasoning capabilities of LLMs. Our findings highlight the potential of dynamic hidden state manipulation for improving model performance on complex reasoning tasks and suggest avenues for future research.

1 Introduction

The field of natural language processing (NLP) has been profoundly transformed by the advent of large language models (LLMs) such as BERT (Devlin et al., 2019), GPT-3 (Brown et al., 2020), and LLaMA (Touvron et al., 2023). These models have set new benchmarks across a variety of tasks, significantly improving our ability to understand and generate human language. Despite these advancements, LLMs continue to face challenges in handling complex reasoning tasks, especially multi-hop questions, which require synthesizing information from multiple sources to arrive at a correct answer.

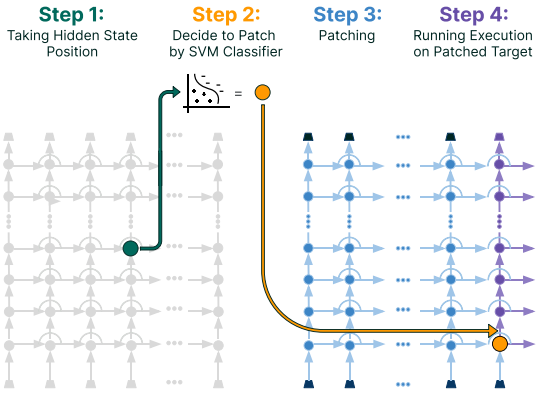

Figure 1: Illustration of the Auto Patch framework used for enhancing multi-hop reasoning in language models. The process involves the following steps: **Step 1**: Select the hidden state from the source layer that will be considered for patching. **Step 2**: The SVM classifier evaluates whether the selected hidden state should be patched. **Step 3**: If patching is needed, the hidden state is transferred to the same position in the target layer. **Step 4**: Execute the forward pass on the patched target, incorporating the patched hidden state into the final output generation. This dynamic adjustment improves the model's ability to reason across multiple steps.

Multi-hop reasoning involves answering questions that necessitate linking disparate pieces of information across different segments. For instance, a question might require first identifying a piece of data and then using that information to find the final answer. Traditional approaches to improving LLMs' performance on such tasks include:

1. **Chain-of-Thought (CoT) Prompting**: This method guides the model to generate explicit intermediate reasoning steps. CoT prompting helps break down complex problems into man-

[3]Code is publicly available at https://github.com/omertalmi5/Auto-Patching.

ageable steps, which significantly enhances performance on tasks like arithmetic and commonsense reasoning. However, CoT often demands extensive fine-tuning and precise prompt engineering, which can be computationally expensive and may not generalize well without considerable manual effort (Wei et al., 2023), (Zhang et al., 2022).

2. **Memory-Augmented Networks**: These networks integrate external memory to store and retrieve information across multiple hops. While they are effective in handling complex reasoning by leveraging stored data, managing this memory efficiently can be challenging. The added computational overhead and complexity can limit their scalability and overall performance (Wang et al., 2023).

3. **Graph-Based Approaches**: These methods use graph neural networks to represent and reason over relationships in the question and relevant information. Graph-based approaches excel in capturing relationships between entities and steps in multi-hop questions. Nevertheless, they are often difficult to scale and seamlessly integrate with LLMs, requiring specialized infrastructure and significant computational resources (De Cao et al., 2019).

Despite their success, these methods come with limitations such as increased computational demands, complexity in implementation, and sometimes only modest performance gains. This study explores how the reasoning capabilities of LLMs can be enhanced through a novel approach inspired by PatchScopes (Ghandeharioun et al., 2024). PatchScopes is a modular framework that allows for inspecting and manipulating hidden states in language models. By "patching" or adjusting these hidden states during the model's computation, PatchScopes can refine internal representations and improve reasoning capabilities.

Objective

In this paper, we aim to enhance the performance of the LLaMA 2 model (7B version) on 2-hop question answering tasks. Leveraging the PatchScopes framework, we introduce the Auto Patch method, which dynamically adjusts hidden states during the model's inference process. Our objective is to improve the model's ability to effectively connect information across multiple reasoning steps.

Contribution

We present a novel approach that utilizes a classifier to determine which hidden states should be patched to enhance the model's performance. By selectively patching these states, we achieved significant improvements over the baseline in handling 2-hop questions. This research underscores the practical application of dynamic hidden state manipulation via PatchScopes, providing insights into optimizing LLM performance and advancing model interpretability.

2 Related Work

Mechanistic Interpretability

Mechanistic interpretability (MI) aims to unravel complex machine learning models by reverse engineering their internal mechanisms into human-understandable algorithms (Geiger et al., 2022; Olah et al., 2018). Understanding these mechanisms allows us to better identify and fix model errors (Vig et al., 2020), steer model outputs (Li et al., 2023), and explain emergent behaviors (Matton and de Oliveira, 2019; Barak et al., 2022). Our work builds on the MI framework by applying dynamic hidden state adjustments, which are informed by the insights gained from model interpretability, to enhance reasoning capabilities.

Activation Patching and Causal Tracing

Activation patching, also known as causal tracing or interchange intervention, is a pivotal technique in MI that helps localize important components within models by intervening on their activations. This method involves modifying specific parts of a model's computation while observing the effect on the output, thus identifying which parts are crucial for certain decisions (Vig et al., 2020; Meng et al., 2022). Common approaches include replacing activations of a corrupted input with those from a clean input and measuring the change in performance (Ghandeharioun et al., 2024). This technique has been extensively used to understand how models store and process factual information and to uncover the computational circuits responsible for specific tasks (Geva et al., 2023).

Multi-Hop Reasoning in Language Models

Multi-hop reasoning requires models to synthesize information from multiple steps or data points to answer a query. This is challenging for large language models (LLMs) as it demands maintaining a coherent logical flow throughout the reasoning process. Approaches to address these challenges include:

1. **Chain-of-Thought (CoT) Prompting**: This technique encourages models to explicitly articulate intermediate reasoning steps, which can significantly improve their performance on tasks requiring structured thinking (Wei et al., 2023; Zhang et al., 2022). CoT prompting is particularly effective for complex, multi-step problems but often requires substantial fine-tuning and careful prompt design.

2. **Memory-Augmented Networks**: These networks utilize external memory to store and retrieve relevant information across multiple reasoning steps. While effective, integrating memory modules adds complexity and computational overhead, which can hinder scalability (Wang et al., 2023).

3. **Graph-Based Approaches**: Graph neural networks (GNNs) model relationships between entities and reasoning steps, excelling in capturing relational data. However, their computational demands and need for specialized infrastructure can make them difficult to scale and integrate with LLMs (Cao et al., 2019).

Despite their strengths, these methods can be limited by their complexity and resource demands. Our Auto Patch method simplifies the enhancement of multi-hop reasoning by dynamically adjusting hidden states during inference, providing a scalable and efficient solution.

Advancing Patchscopes with Auto Patch

Patchscopes provide a framework for inspecting and intervening on hidden states within LLMs, allowing researchers to refine model behaviors and improve reasoning capabilities. This method has shown promise in correcting errors and enhancing interpretability by directly manipulating the model's internal representations during computation (Ghandeharioun et al., 2024). However, traditional Patchscopes often require manual separation of prompts into their components, making them less practical for real-world applications.

Our work extends Patchscopes by automating the patching process through the Auto Patch method. By using classifiers to dynamically determine which hidden states to patch, Auto Patch simplifies the intervention process, making it more suitable for practical deployment. This automated approach enhances the scalability and efficiency of hidden state adjustments during inference, providing a robust solution for multi-hop reasoning tasks in LLMs.

3 Methodology

This section details the improved methods used to enhance the multi-hop reasoning abilities of the LLaMA 2 model by integrating an automated patching process within the PatchScopes framework. Our approach removes the need for manually separating prompts into components, making it more practical for real-world use.

3.1 Advancing PatchScopes with Auto Patch

PatchScopes is a key framework for exploring and adjusting hidden states within large language models (LLMs), helping to refine model behaviors and improve reasoning capabilities. Although promising, traditional PatchScopes often require extensive manual work, which limits their use outside of research settings. Our development extends this framework through the Auto Patch method, which uses classifiers to automatically decide which hidden states to patch. This not only makes the patching process easier but also makes the model more scalable and efficient, especially in complex multi-hop reasoning tasks. The main idea behind our method is that the decision to apply a patch can be effectively learned and optimized, thus improving the model's accuracy and functionality.

3.2 Data Collection and Preprocessing

To prepare our dataset for evaluating multi-hop reasoning capabilities, we followed a systematic process with each multi-hop question from the MuSiQue dataset:

1. **Initial Model Run:** We first ran the model using each prompt to record all hidden states at a specific layer.

2. **Patching Hidden States:** Subsequently, we conducted another model run where we

patched selected hidden states into earlier layers.

3. **Performance Analysis:** We then carefully analyzed how each patch affected the model's solve rate.

4. **Data Recording:** For each prompt, we recorded the original hidden state, the prompt itself, the layer involved, and the observed improvement in solve rate.

Each entry in our dataset thus includes comprehensive details that provide a solid foundation for the further training of our classifier, ensuring a systematic approach to data collection and preprocessing.

3.3 Classifier Training for Auto Patching

To support the automatic patching process, we trained a Support Vector Machine (SVM) classifier using a specific kernel to improve decision-making. Training involved extracting features from interactions between various layers, where hidden states from higher layers were experimentally replaced with those from lower layers to measure their impact on performance. This method allowed us to train the SVM classifier effectively, enabling it to dynamically determine the best positions for patching during model inference, thus removing the need for manual layer specification and focusing on enhancing reasoning through learned interventions.

3.4 Auto Patching Framework Implementation

At the heart of our methodology, the Auto Patching Framework dramatically changes how hidden states are manipulated during model inference. Initially, the model processes the prompt through a standard forward pass to capture the baseline hidden states. These states are then evaluated by the trained classifier to decide if and where patches are needed. If patching is beneficial, the model undergoes a second forward pass where selected hidden states are adjusted as recommended by the classifier. This dual-phase operation not only maintains the necessary context for accurate multi-hop reasoning but also adds a dynamic, learnable element to hidden state manipulation, greatly improving the model's performance on various prompts.

To enable this process, we developed specific patching code that started with manual patching to gain initial insights into the strategic manipulation of hidden states. This phase was crucial for determining the impactful interactions between layers and setting the groundwork for automated interventions.

Following these initial insights, we progressed to automating the patching process. The decision-making capabilities of our SVM classifier were seamlessly integrated into the model's inference engine, enabling dynamic and automatic patching during the question-answering process. This integration ensures that the model can adjust hidden states in real-time, based on classifier predictions, optimizing performance without the need for manual intervention.

This comprehensive development and integration of patching code into the Auto Patching Framework not only streamlines the entire process but also enhances the model's ability to handle complex reasoning tasks more efficiently.

4 Experiments

This section presents the experiments conducted to evaluate the effectiveness of our Auto Patch method. We provide a detailed description of the experimental setup, evaluation metrics, dataset characteristics, results, and comparative analysis with baseline and Chain-of-Thought (CoT) methods.

4.1 Evaluation Metrics

To evaluate the performance of our method, we utilized solve rate metric:

- **Solve Rate**: The primary metric was the solve rate, defined as the proportion of correctly answered 2-hop questions out of the total questions.

4.2 Dataset Description

Our dataset contains 24,912 samples created from 1,024 prompts, each representing a different execution of the Patchcopes method with patching from and to specific positions. For each prompt, we executed from source layer 15 to target layer 8, for each position in the prompt. The main features of each sample are: 'prompt_source' and 'prompt_target', the same and consist of a full two-hop question; 'position_source' and 'position_target', the same position that from and to; 'hop3', the correct answer to the two-hop question; 'generations_patched', the LLM's response to the two-hop question; 'is_correct_patched', a boolean

value indicating if the LLM generated the correct answer; and 'hidden_rep', the hidden state patched during execution, represented as a float array of length 4096. 'hidden_rep' is the input for the classifier, and 'is_correct_patched' is the label. 23% of the samples have 'is_correct_patched' as true. This does not mean that the solve rate is 23%, because long prompts occur in more samples.

4.3 The Experiment Flow

We receive 1,024 two-hop questions without any separation into hops. We pass them through the first inference in the LLM. Then, each hidden state from layer 15 passes through the classifier. If the classifier returns true, in the second inference, we replace it with the hidden state in the same position in layer 8. We then complete the second inference and check if it returns the correct answer.

4.4 Results and Analysis

The results of our experiments demonstrate the impact of the Auto Patch method on model performance compared to the baseline and Chain-of-Thought (CoT) methods. Table 1 summarizes the solve rate of each approach on the 2-hop questions from the MuSiQue dataset.

Method	Solve Rate (%)
Baseline (LLaMA 2)	18.45
Chain-of-Thought (CoT)	27.44
Auto Patch (Ours)	23.63

Table 1: Comparison of solve rate on MuSiQue 2-hop questions.

Auto Patch (Ours): Our Auto Patch method achieved a solve rate of 23.63%, significantly outperforming the baseline but not surpassing the CoT method.

To determine optimal positions for patching, we employed an SVM classifier from the sklearn library, utilizing a Radial Basis Function (RBF) kernel. Given the imbalanced nature of the dataset, we applied the SMOTETomek method for balancing, which involved both sampling and generating the minority class. Standardization was also part of the preprocessing to enhance the classifier's performance.

The classifier achieved an accuracy of 0.81. Table 2 provides a detailed classification report for the SVM classifier's performance at the source layer (layer 15).

	Precision	Recall	F1-Score	Support
False	0.84	0.92	0.88	3808
True	0.64	0.45	0.53	1175
Accuracy		0.81		
Macro Avg	0.74	0.69	0.70	4983
Weighted Avg	0.80	0.81	0.80	4983

Table 2: Classification Report for the classifier

As shown in Table 2, The accuracy of the classifier was 0.81. While the overall precision and recall on the data after the preprocessing suggest acceptable performance. The classifier performed with high precision and recall for the 'False' class compared to the 'True' class, the majority class ('False') is predicted more accurately. On the real data, the model predicts *True* for almost all positions. All samples have one or two positions that are not patched. Most of positions that not patched, their tokens are: start of sentence symbol <s> (can come with the real prompt after it and many <unk> before it), or unknown symbol <unk> or single dot "." , as shown in Figure 2. The classifier learns the structure of the sentence and mostly does not patch the first token. Intuitively, we can view this as patching only the first token from layer 8 to layer 13. The intuition for patching most of the positions is that the second hop should influence the first hop in our dataset.

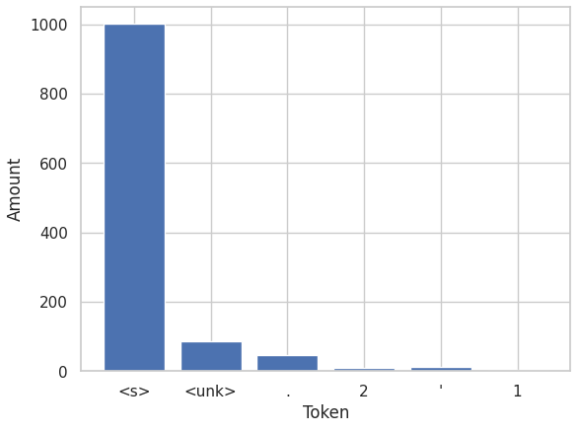

Figure 2: Histogram showing the tokens of not patched position number.

4.5 Discussion

The effectiveness of the Auto Patch method can be attributed to several key factors that enhance the multi-hop reasoning capabilities of large language models (LLMs).

Intuition Behind Patching The Auto Patch method uses dynamic hidden state patching to reintroduce and modify information at specific layers during inference. This process refines the model's understanding, maintaining coherence and context across reasoning steps of the different hopes. It allows the model to link separate pieces of information, which is essential for multi-hop reasoning tasks.

Increased Computational Utilization Similar to Chain-of-Thought (CoT) prompting, which utilizes more compute by guiding the model through intermediate reasoning steps, the Auto Patch method enhances computational engagement. By dynamically adjusting hidden states, the method directs computational resources towards synthesizing complex, multi-step information, improving accuracy and depth of reasoning.

Classifier's Role in Effective Patching The integration of a Support Vector Machine (SVM) classifier is crucial in determining when and where patches should be applied. The classifier identifies scenarios where patching hidden states is most beneficial, allowing dynamic adaptation and improvement of internal representations. This showcases a learned strategy for enhancing multi-hop reasoning.

Comparison with Random Classification The classifier's effectiveness is highlighted by its performance compared to a random classification baseline. The random classification serves as a control, demonstrating that the observed improvements are due to the targeted, learned patching strategy rather than arbitrary interventions. This comparison underscores the classifier's role as a learned, intelligent component in the Auto Patch framework.

Optimal Layer Selection for Patching The selection of layers for patching, particularly transferring information from higher (15th) to lower (8th) layers, aligns with the intuition that early layers capture more basic and simple knowledge. These layers encode the basic representations crucial for maintaining context in multi-hop reasoning. Strategic injection of refined information into these early stages enhances the model's ability to integrate complex information, improving performance on multi-hop questions.

Overall, the Auto Patch method's design and execution reflect a sophisticated approach to leveraging dynamic hidden state manipulation to bolster LLMs' reasoning capabilities. The combination of learned patching strategies and targeted computational enhancements offers a promising direction for advancing multi-hop reasoning in language models.

4.6 Unsuccessful Experiments

In the course of developing and evaluating the Auto Patch method, several experiments did not yield the anticipated improvements. These insights are crucial for guiding future research and refining our approach.

Positional Patch Adjustments One explored avenue was concatenating the position number to the hidden state that was fed to the classifier. The hypothesis was that the position number might add information to the classifier about the structure of the sentence. However, these experiments showed the same results. The suggestion is that the hidden states in LLaMA2 already incorporate positional information through the initial addition of position embeddings to the token embeddings. This integration means that the positional context is inherently preserved in the hidden states, rendering the explicit addition of position numbers unnecessary for the classifier.

Random Classification To validate the necessity and effectiveness of the learned classifier, experiments were conducted where hidden states were patched based on random classification. This approach served as a control against the SVM classifier's performance. Results from random patches were sporadic and did not yield meaningful improvements in solve rates. This outcome confirms the importance of the classifier's learned decision-making in identifying beneficial patching opportunities, emphasizing that targeted, informed interventions are essential for optimizing multi-hop reasoning.

These unsuccessful experiments underscore the complexity of dynamic hidden state manipulation and highlight the need for strategic, informed decision-making processes to enhance LLM performance.

5 Conclusion

In this paper, we introduced the Auto Patch method, designed to enhance multi-hop reasoning in large language models by dynamically adjusting hidden

states. Our results on the MuSiQue dataset show improved performance on 2-hop questions.

Despite these improvements, the method has limitations. The current classifier's focus on isolated hidden states overlooks broader context. Future work could explore context-aware classifiers that incorporate neighboring states for better decision-making. Additionally, applying patches to the same positions across layers may not be optimal; developing adaptive strategies for layer and position selection could enhance effectiveness. Finally, expanding testing to a broader range of datasets and question types will be crucial to evaluate the method's broader applicability.

Auto Patch demonstrates potential in advancing the reasoning capabilities of language models, suggesting several avenues for further exploration and refinement.

Acknowledgements

We would like to express our deepest gratitude to Dr. Mor Geva, Maor Ivgi, and Daniela Gottesman for their invaluable guidance and support throughout this project. Their insights and expertise have significantly contributed to the successful completion of our research.

Limitations

While the Auto Patch method improves multi-hop reasoning, it has notable limitations. First, the SVM classifier's decisions are based on isolated hidden states, which may ignore broader context. Second, the method applies patches to identical positions across layers, limiting flexibility and potential optimization for different tasks.

References

Boaz Barak, David J. Schwab, and Greg Ver Steeg. 2022. Emergent behaviors in large neural networks.

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

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2019. Question answering by reasoning across documents with graph convolutional networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2019. Question answering by reasoning across documents with graph convolutional networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2306–2317, Minneapolis, Minnesota. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Atticus Geiger, Zhengxuan Wu, Hanson Lu, Josh Rozner, Elisa Kreiss, Thomas Icard, Noah D. Goodman, and Christopher Potts. 2022. Inducing causal structure for interpretable neural networks.

Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. 2023. Dissecting recall of factual associations in auto-regressive language models.

Asma Ghandeharioun, Avi Caciularu, Adam Pearce, Lucas Dixon, and Mor Geva. 2024. Patchscopes: A unifying framework for inspecting hidden representations of language models.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. 2023. Inference-time intervention: Eliciting truthful answers from a language model.

Alexandre Matton and Luke de Oliveira. 2019. Emergent properties of finetuned language representation models.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt.

Christopher Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. 2018. The building blocks of interpretability. *Distill*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Jesse Vig, Kevin Yang, Jasmijn Bastings, Vinodkumar Prabhakaran, Been Kim, Dmytro Karamshuk, and Chris E Develder. 2020. Causal mediation analysis for interpreting neural nlp: The case of gender bias. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Boshi Wang, Sewon Min, Xiang Deng, Jiaming Shen, You Wu, Luke Zettlemoyer, and Huan Sun. 2023. Towards understanding chain-of-thought prompting: An empirical study of what matters. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2717–2739, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models.