Autonomous Prompting: Determining Optimal Prompting Strategies for Large Language Models

Henry Yi[1], **Ishaan Singh**[1], **Saharsh Barve**[1], **Veda Kailasam**[1]
[1]University of Illinois, Urbana-Champaign
{weigang2, is14, ssbarve2, vedak2}@illinois.edu

Abstract

The rapid advancement in Large Language Models (LLMs) necessitates innovative approaches to optimize their application across diverse tasks. This project introduces an autonomous system designed to dynamically select the most appropriate prompting strategy for LLMs (from zero-shot, few-shot, chain-of-thought, self-consistency with chain-of-thought), thus maximizing their efficiency and applicability in real-world scenarios. Our system evaluates and adapts prompting strategies in real-time, ensuring optimal interaction between LLMs and human queries. We test the efficacy of our approach on a subset of the GSM8K dataset, which allows us to assess the system's performance on complex mathematical reasoning tasks. The code for this project is available at https://github.com/saharsh1005/autonomous-prompting.

1 Introduction

With significant advancements in the field of Large Language Models (LLMs) and their expanding roles in a wide variety of tasks, prompting has become an indispensable technique to guide these powerful models accurately. It acts as a critical interface with both Large Language and Vision models (LLMs and LLVMs), where specific inputs are crucial for steering their outputs. This technique of transforming human queries into targeted inputs to influence model responses is central to their function. By crafting precise prompts, the performance of these models is significantly enhanced. The strategic design of prompts, known as **prompt engineering**, is vital because it not only optimizes model outputs but also ensures that these complex systems can be effectively applied in real-world scenarios, bridging the gap between advanced technology and practical usability.

Various prompting strategies have been developed recently to optimize the performance of LLMs. Each of these strategies offers unique advantages and is suitable for different scenarios. **Zero-Shot Prompting** employs the model's pre-existing knowledge to respond to queries without any additional examples or context. This approach is especially useful for general and broad inquiries where specialized input-output mapping examples are not provided alongside the task description. Unlike zero-shot prompting, **Few-Shot Prompting** involves providing the model with a small set of examples to guide its response. This helps tailor the model's outputs towards a specific task or application. Providing a few high-quality examples has proven to improve model performance on complex requests but also requires more input tokens. **Chain of Thought (CoT)** helps the model perform complex reasoning tasks by encouraging it to process and articulate its intermediate reasoning steps before arriving at the final answer. The simple yet powerful directive, "Let's think step by step," considerably boosts the model's ability to tackle intricate problems, thereby improving its performance for such tasks. **Self-Consistency with CoT** involves iterating the CoT-enhanced question multiple times, collecting the answers, and then using a voting mechanism to select the most common response. This method improves overall decision-making accuracy by emphasizing consistent responses. However, this approach also introduces computational inefficiencies, as the need for multiple iterations can lead to increased processing time and operational costs.

Given the complexity and diversity of tasks for which LLMs are utilized, a dynamic and adaptable prompting strategy is essential. Currently, the selection of the most effective prompting strategy tends to be manual and heuristic-based, which constrains the scalability and operational efficiency of LLM applications. Our project, Autonomous Prompting, aims to improve this process by automating the selection of optimal prompting strategies using machine learning. This method assesses the nature of the query and its contextual aspects to tailor the strategy, thereby enhancing the practicality and effectiveness of LLM deployments across various domains and ensuring efficient and highly effective use of resources.

Our work is centered around four principal dimensions:

1. **Hierarchical Prompt Strategy Selection:** We have developed a systematic methodology to assess and choose the most suitable prompting strategies, ranging from simple to complex, depending on the query's specifics.
2. **Complex Dataset Utilization:** We use a subset of the GSM8K dataset to test and verify the efficiency of autonomous prompting.
3. **Dynamic Strategy Adaptation:** Our approach employs a search-based retriever and a reranking cost model to efficiently select the optimal prompting strategy for each query.
4. **Performance Benchmarking:** We benchmark our system using the LLaMA-8B model and measure its performance in comparison to Chain-of-Thought and Self-Consistency. Our goal is to show that we can achieve similar, if not better, results while being less computationally expensive.

2 Related Work

The evolution of Large Language Models in recent years has given rise to diverse prompting techniques, enhancing their performance on a wide variety of applications; from straight-forward text generation to complex reasoning and decision making. Early work by [Radford et al., 2019] [Brown et al., 2020] demonstrated model performance using zero-shot and few-shot approaches and how in the case of few-shot prompting, the model could perform well when it was given a few input-output mappings to build a context thus removing the dependence on task-specific fine-tuning. Since then, zero-shot and few-shot approaches have been explored in a wide variety of tasks [Schick and Schütze, 2020, Raffel et al., 2019, Arora et al., 2022, Jiang et al., 2023].

The chain-of-thought (CoT) prompting technique, as introduced by [Wei et al., 2023], has played a pivotal role in advancing the reasoning capabilities of LLMs. This technique encourages models to generate intermediate steps of reasoning, thus enabling them to tackle complex reasoning tasks in an effective manner. Building upon this foundation, [Kojima et al., 2023] extended CoT prompting approach by demonstrating its effectiveness in a zero-shot setting, eliminating the need to provide explicit examples of reasoning, thus showcasing the underlying reasoning potential of LLMs. Furthermore, [Wang et al., 2023] introduced the self-consistency methodology, which leverages multiple iterations of task processing. By generating diverse reasoning paths and converging on the most consistent solution (similar to ensemble approach in machine learning), this method bolsters decision-making abilities of language models, further solidifying the impact of CoT prompting on the evolution of LLM capabilities.

To address the limitations of outdated training data for LLMs and to ensure a model can access up-to-date information when solving a user query, Retrieval-Augmented Generation (RAG) [Lewis et al., 2021] technique was proposed.[Guu et al., 2020] further explores the use of retrieval mechanisms in enhancing language model performance by dynamically selecting relevant information. Building upon this, our approach of search-based retriever to select optimal strategy follows this technique but applies it at prompt engineering level for strategy selection.

Dynamic prompting techniques that automatically adjust the prompt based on the task at hand have gained considerable importance. [Zhang et al., 2022] introduced the Auto-CoT (Automatic Chain-of-Thought) approach, which generates reasoning chains by sampling questions, enabling more efficient and effective task-solving. Along these lines, [Shao et al., 2023] proposed synthetic prompting approach for a model to generate chain of thought demonstrations for itself, reducing the burden of manually creating examples by hand. Furthermore, the ReAct framework, introduced by Yao et al. [Yao et al., 2023], extends dynamic prompting by combining reasoning with actions, allowing the

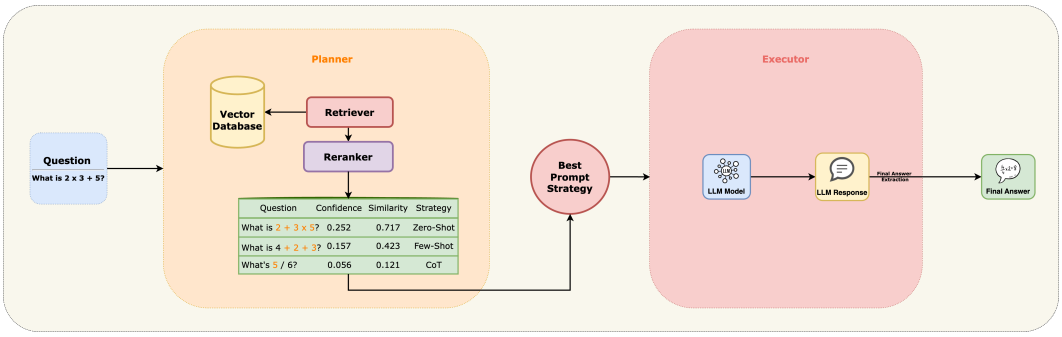

Figure 1: Autonomous Prompt Selection Pipeline

model to make decisions and interact with its environment more effectively. Additionally, [Do et al., 2024] proposed an approach for automatically selecting optimal prompts by first clustering training data, generating candidate prompts and training a prompt evaluator which gets used to select the best prompt at test time.

In summary, the prompt engineering landscape has undergone significant advancements, progressing from zero-shot and few-shot methods to more sophisticated techniques like chain-of-thought prompting, which facilitates complex reasoning tasks. Additionally, dynamic and adaptive prompting strategies have been developed to minimize reliance on manually crafted examples. Overall, these advancements bolster the use of LLMs across wide range of tasks in real-world scenarios.

3 Autonomous Prompt Selection

We propose an **Autonomous Prompt Selection** (Figure 1) framework that automates the process of creating, evaluating, and selecting optimal prompts for LLMs. It consists of three key components: Dataset Creation, where diverse prompting strategies are systematically created and evaluated; Planner Agent: Knowledge Base Construction, which involves generating embeddings and retrieving relevant strategies; and Executor Agent: Strategy Execution and Optimization, which dynamically applies and executes the most effective prompt strategy for a given query. And in the end, an Auto Prompt Agent is created to orchestrate the end-to-end pipeline to ensures adaptability, efficiency, and scalability across tasks.

3.1 Dataset Creation

The dataset creation process is a critical foundational step in the Autonomous Prompt Selection framework. This stage focuses on systematically applying multiple prompting strategies to a mathematical reasoning benchmark dataset, ensuring a comprehensive representation of diverse problem-solving approaches. The resultant dataset is pivotal for training and evaluating the pipeline's embedding, retrieval, and re-ranking mechanisms.

The objective of the dataset creation process is to construct a rich and structured dataset that evaluates the performance of various prompting strategies across mathematical reasoning problems. By systematically generating model responses using different strategies and capturing the associated metadata, the dataset enables a robust comparison of strategies, balancing accuracy and computational efficiency.

Source Dataset: We leverage the GSM8K dataset, a benchmark dataset specifically designed for evaluating mathematical reasoning capabilities in Large Language Models (LLMs). The dataset contains mathematically challenging word problems that require reasoning and computation to arrive at the correct answer. From the GSM8K test split, a subset of 727 questions is selected for processing. These questions provide a representative sample of diverse problem types to evaluate the effectiveness of prompting strategies.

Each question in the GSM8K subset is processed using four distinct prompting strategies: Zero-Shot, Few-Shot, CoT, Self-Consistency with CoT. For each question, the following steps are performed:

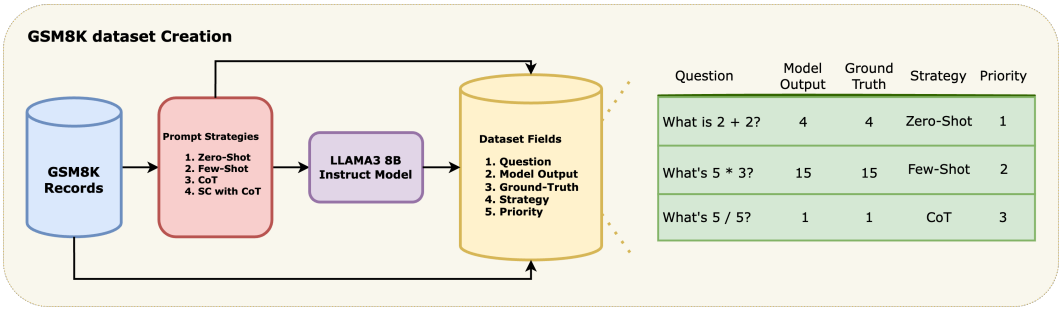

Figure 2: Autonomous Prompt Generated Dataset

1. **Prompt Generation:** A prompt is dynamically generated based on the chosen strategy. For example, in the CoT strategy, the question is prefixed with instructions encouraging step-by-step reasoning.
2. **Model Querying:** The generated prompt is sent to the **LLaMA3 8B Instruct Model** to produce a response. The model's output, referred to as the **generated answer**, is recorded.
3. **Ground Truth Validation:** The **correct answer** is extracted from the ground truth annotations in GSM8K. The model's generated answer is compared to this correct answer to determine accuracy.
4. **Metadata Assignment:** Each strategy is assigned a priority based on its computational cost:
   - Zero-shot: Priority 0 (minimal computation).
   - Few-shot: Priority 1.
   - Chain-of-Thought: Priority 2.
   - Self-Consistency with CoT: Priority 3 (highest computation due to multiple output generations).

The final dataset (Figure 2) contains $727 \times 4 = 2908$ rows, representing the evaluation of each question across all four strategies. Each row in the dataset encapsulates the interplay between problem complexity, reasoning strategy, and computational cost. An example entry is as follows:

```
Question: Sandra's neighbor gives her a basket of 9 eggs
every time she babysits their daughter. To make a Spanish
flan, she needs 3 eggs. If Sandra has been tasked to make
15 Spanish flans for her school fundraiser, how many times
does Sandra have to babysit? Generated Answer: 5; Correct
Answer:5; Strategy:Zero-Shot; Priority:0.
```

3.2 Planner Agent: Knowledge Base Construction

The Planner Agent (Figure 3) serves as the core decision-making component of the Autonomous Prompt Selection framework. Its primary role is to analyze incoming queries, leverage a prebuilt knowledge base to find relevant examples, and recommend the optimal prompting strategy for the given problem. This process encompasses constructing an efficient knowledge base, retrieving relevant data, and re-ranking retrieved results to identify the best match. The Planner Agent comprises three integral sub-components: Embedding generation and storage, Retrieval, and Re-ranking.

The foundation of the Planner Agent lies in the creation of a robust knowledge base (Figure 4). This process entails embedding questions from the dataset and storing them in a vector database for efficient retrieval. The following steps detail the knowledge base construction:

1. **Embedding Generation:** Each question in the dataset is transformed into an embedding vector (`dim=1024`) using the Cohere (*embed-english-v3.0*) via the Cohere API.
2. **Metadata Assignment:** Metadata associated with each question is stored alongside the embeddings to provide context and aid retrieval. This metadata includes: `question`, `prompt strategy`, `priority`

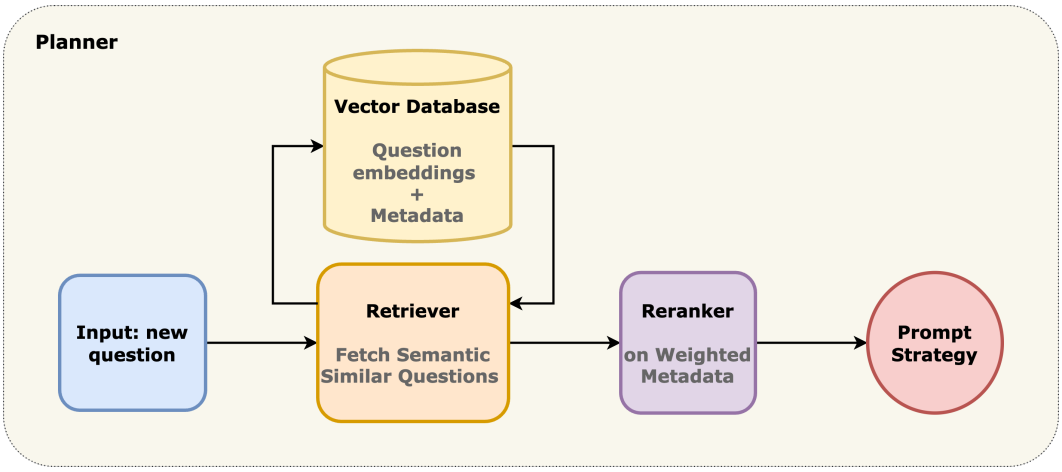

Figure 3: Planner Agent

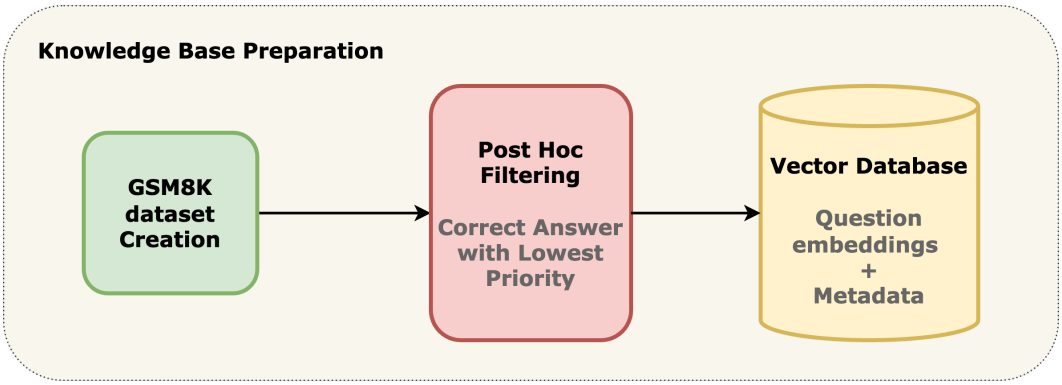

Figure 4: Knowledge Base Creation

3. **Vector Database Storage:** The embeddings and metadata are stored in a Pinecone vector database, which facilitates fast and scalable similarity searches. Pinecone's cosine similarity metric is used to measure the relevance of stored entries during retrieval.

3.2.1 Retrieval

Once the knowledge base is constructed, the Planner Agent retrieves the most relevant questions for a new query. The retrieval process involves embedding the query and performing a similarity search against the stored embeddings in the vector database. The steps are as follows:

1. **Query Embedding:** The incoming query is embedded using the same Cohere model employed during knowledge base construction. This ensures consistency in vector representations.
2. **Similarity Search:** The query embedding is compared against stored embeddings in the Pinecone index. A top-K similarity search retrieves the most semantically relevant questions. Each retrieved entry includes: `original question, prompt strategy, priority, relevance score`
3. **Preliminary Results:** The retrieved results yield a ranked list of potential matches, which are passed to the re-ranking mechanism for further refinement.

3.2.2 Re-Ranker

The re-ranking step refines the list of retrieved results using a confidence-based scoring function to identify the most effective prompting strategy:

$$C(r) = \min\left(\frac{(p_r + s_r)}{p_{max} + 1} \times \beta_r, 1.0\right)$$

where:

- $C(r)$ is the final confidence score for result r
- p_r is the priority of the strategy (0-3)
- s_r is the cosine similarity score (0-1)
- p_{max} is the maximum priority value (3)
- β_r is the strategy-specific multiplier:

$$\beta_r = \begin{cases} 1.15 & \text{if strategy is SC-CoT} \\ 1.0 & \text{otherwise} \end{cases}$$

The denominator $(p_{max} + 1)$ normalizes the sum of priority and similarity scores to a [0,1] range, since the maximum possible value is the sum of highest priority (3) and highest similarity (1). Results are sorted in descending order of $C(r)$, with the highest confidence score determining the selected strategy. This formula balances semantic relevance (via similarity score), computational efficiency (via priority), and reasoning reliability (via strategy-specific adjustment) to optimize strategy selection. By relying on this sorted list, the Planner Agent dynamically selects the most effective strategy tailored to the problem at hand, ensuring that the system adapts seamlessly.

3.2.3 Strategy Recommendation

The final output of the Planner Agent is the recommended strategy and a corresponding prompt for the Executor Agent. The steps for generating the strategy recommendation are as follows:

1. Strategy Selection: Based on the re-ranked results, the strategy with the highest confidence is selected. If no entries are found, the Planner defaults to a zero-shot strategy as a fallback.
2. Prompt Generation: The Planner generates a prompt using the selected strategy for the question.
3. Output Delivery: The Planner outputs and forwards the final prompt and strategy to the Executor for execution.

3.3 Executor Agent: Strategy Execution and Optimization

The Executor Agent is the operational core of the Autonomous Prompt Selection framework, responsible for executing the selected prompt strategy and generating the final answer for the given query. It collaborates closely with the Planner Agent to ensure effective implementation of the optimal strategy while maintaining high-quality outputs.

The Executor Agent executes prompts generated by the Planner Agent using the specified strategy. Each strategy is designed to optimize the reasoning capabilities of the Large Language Model (LLM) for the given query. For SC-CoT, the Executor generates multiple response samples from the LLM, each representing a different reasoning pathway. A majority voting mechanism is applied to select the most common answer across all generated samples. This method ensures robustness and reliability by leveraging consensus-based reasoning. For other strategies such as Zero-shot, Few-shot, and Chain-of-Thought (CoT), the Executor generates a single response directly based on the provided prompt. The output is parsed to extract the final answer, ensuring adherence to the required format.

The Executor Agent bridges the gap between strategy selection and practical implementation within the framework. By dynamically adapting to different strategies, such as SC-CoT and Few-shot, and leveraging LLM capabilities efficiently, the Executor ensures high-quality responses for diverse queries. Its robust design enhances the framework's ability to handle complex reasoning tasks while maintaining efficiency and adaptability.

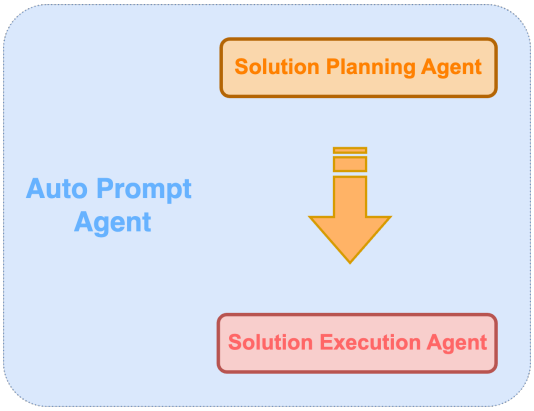

Figure 5: Knowledge Base Creation

3.4 Ensemble Auto Prompt Agent: Orchestrating Solution Planning and Solution Execution

The Auto Prompt Agent (Figure 5) is the top-level agent that coordinates the Solution Planning (Planner) Agent and the Solution Execution (Executor) Agent. It first takes in a user's question and feeds that into the Planner Agent. Then, it puts the prompt adapted to the best strategy by the Planner Agent to the Executor Agent for execution. Consequentially, it would output the final answer extracted by the Executor Agent and present it back to the user. Auto Prompt Agent encapsulate the entire workflow and works as an end-to-end solution to the problem concerned in the first place.

4 Results and Evaluation

Our experimental evaluation focused on two key aspects: (1) the comparative effectiveness of different prompting strategies on the GSM8K dataset, and (2) the performance of our AutoPrompt system against established baselines. Table 1 presents the results of our initial strategy evaluation.

Strategy	**Accuracy (%)**
Zero-shot	43.74
Few-shot	71.80
Chain-of-thought	70.70
Self-consistency-COT	78.40

Table 1: Comparison of prompting strategies on GSM8K test dataset (n=727)

The initial evaluation demonstrates the clear superiority of more sophisticated prompting strategies over simple zero-shot approaches, with self-consistency chain-of-thought (SC-COT) showing the highest accuracy at 78.40%. However, this preliminary analysis does not account for computational costs and runtime efficiency, which are crucial factors in practical applications.

To evaluate our AutoPrompt system's effectiveness, we conducted a comprehensive comparison against the two strongest baseline approaches: standard chain-of-thought (COT) and self-consistency chain-of-thought (SC-COT). Table 2 presents these results.

Strategy	**Accuracy (%)**	**Avg Tokens**	**Runtime (s)**
AutoPrompt	**89.00**	**399.03**	**3.21**
Chain-of-thought	79.00	489.06	3.86
Self-consistency-COT	88.00	2445.30	16.78

Table 2: Performance comparison on GSM8K test dataset (last 100 rows)

Our AutoPrompt system achieves remarkable results, demonstrating superior performance across all key metrics. Most notably, it achieves the highest accuracy (89%) while maintaining significantly lower computational costs than SC-COT. The system requires only 399.03 tokens per question on

average, compared to SC-COT's 2445.30 tokens, representing an 83.7% reduction in token usage. Similarly, the average runtime per question (3.21 seconds) is comparable to standard COT (3.86 seconds) and dramatically lower than SC-COT (16.78 seconds).

The superior performance of AutoPrompt can be attributed to several key factors:

1. **Dynamic Strategy Selection:** Rather than applying a one-size-fits-all approach, AutoPrompt's confidence-based reranking system allows it to choose the most appropriate strategy for each question. This adaptability enables it to achieve high accuracy while avoiding unnecessary computational overhead.
2. **Efficient Resource Utilization:** By intelligently selecting simpler strategies when appropriate, AutoPrompt maintains the benefits of advanced techniques like SC-COT while significantly reducing average token usage and runtime.
3. **Pattern Recognition:** The embedding-based similarity search, combined with strategy metadata, allows AutoPrompt to leverage patterns in successfully solved problems, effectively learning from past performance to optimize strategy selection.

To validate the robustness of our results, we specifically tested on the last 100 rows of the GSM8K test dataset, ensuring no overlap with the training data used for embedding generation. This separation prevents any potential data leakage and demonstrates the system's generalization to unseen problems.

The substantial reduction in computational resources required by Auto Prompt (83.7% fewer tokens and 80.9% faster runtime compared to SC-COT) while maintaining comparable or superior accuracy represents a significant advancement in practical LLM deployment. These efficiency gains directly translate to reduced operational costs and improved scalability, making sophisticated prompting strategies more accessible for real-world applications.

Furthermore, the system's ability to match or exceed the accuracy of SC-COT (89% vs 88%) while maintaining the computational efficiency of simpler approaches demonstrates successful resolution of the accuracy-efficiency trade-off that has traditionally characterized prompt engineering strategies.

5 Conclusion

The proposed Autonomous Prompting framework introduces a novel approach to optimizing prompt generation and selection for Large Language Models (LLMs). By leveraging a structured pipeline consisting of three key components—Dataset Creation, Planner Agent, and Executor Agent—the system systematically addresses the challenge of aligning diverse strategies to complex queries. This framework not only enhances the reasoning capabilities of LLMs but also minimizes manual intervention, thereby improving scalability and adaptability across a wide range of applications.

The Planner Agent effectively constructs a knowledge base, retrieves relevant strategies, and ranks them based on a confidence function, ensuring the selection of the optimal approach tailored to the problem. The Executor Agent then operationalizes this decision by executing the selected strategy and refining the results for robust and reliable outcomes. Together, these components form a cohesive pipeline that dynamically adapts to different query contexts, maximizing LLM utility and accuracy.

While the current framework demonstrates significant improvements in prompt optimization and execution, several areas offer potential for further enhancement and exploration:

1. Dynamic Strategy Adaptation:Incorporate reinforcement learning techniques to enable real-time strategy adaptation based on the evolving performance of LLMs and user feedback.
2. Introduce a feedback mechanism where end-users can provide insights into the generated answers, helping the system refine its prompts and strategies over time.

The proposed directions for future work aim to further strengthen the versatility and robustness of the framework, aligning it with the rapidly evolving demands of natural language processing tasks. By addressing these challenges, the Autonomous Prompting system can establish itself as a cornerstone for LLM optimization and adaptive reasoning in the AI domain.

References

Simran Arora, Avanika Narayan, Mayee F. Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. Ask me anything: A simple strategy for prompting language models, 2022. URL https://arxiv.org/abs/2210.02441.

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. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Viet-Tung Do, Van-Khanh Hoang, Duy-Hung Nguyen, Shahab Sabahi, Jeff Yang, Hajime Hotta, Minh-Tien Nguyen, and Hung Le. Automatic prompt selection for large language models, 2024. URL https://arxiv.org/abs/2404.02717.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training, 2020. URL https://arxiv.org/abs/2002.08909.

Pengcheng Jiang, Shivam Agarwal, Bowen Jin, Xuan Wang, Jimeng Sun, and Jiawei Han. Text-augmented open knowledge graph completion via pre-trained language models, 2023. URL https://arxiv.org/abs/2305.15597.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023. URL https://arxiv.org/abs/2205.11916.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. URL https://arxiv.org/abs/2005.11401.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019. URL https://api.semanticscholar.org/CorpusID:160025533.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019. URL http://arxiv.org/abs/1910.10683.

Timo Schick and Hinrich Schütze. Exploiting cloze questions for few-shot text classification and natural language inference. *CoRR*, abs/2001.07676, 2020. URL https://arxiv.org/abs/2001.07676.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Synthetic prompting: Generating chain-of-thought demonstrations for large language models, 2023. URL https://arxiv.org/abs/2302.00618.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

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, 2023. URL https://arxiv.org/abs/2201.11903.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models, 2022. URL https://arxiv.org/abs/2210.03493.

Appendix

AutoPrompt Selection Distribution

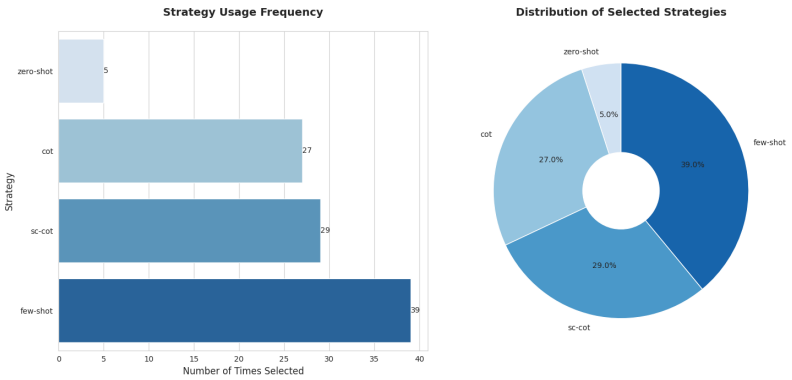

Figure 6: Distribution of strategies chosen by AutoPrompt

The distribution analysis of selected strategies reveals that few-shot learning emerged as the predominant approach (39%), followed by self-consistency chain-of-thought (sc-cot, 29%) and standard chain-of-thought (cot, 27%). Zero-shot learning was employed least frequently (5%). This heterogeneous distribution demonstrates our framework's capability to implement adaptive strategy selection, choosing different approaches based on question requirements rather than defaulting to a single method. The balanced utilization across the three dominant strategies indicates effective discrimination between scenarios where different reasoning approaches may be more advantageous.

AutoPrompt Relative Performance Gain

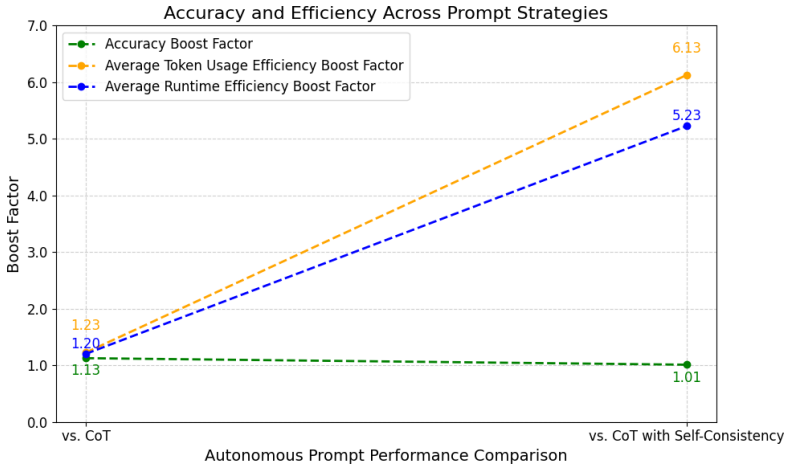

Figure 7: Accuracy and Efficiency Boost using Auto Prompt

To quantify the relative performance gains, we analyzed the boost factors of our autonomous prompting approach compared to both standard Chain-of-Thought (CoT) and CoT with Self-Consistency (SC-CoT) strategies across Accuracy, Token Usage Efficiency, and Runtime Efficiency. When evaluated against CoT, our approach maintains comparable accuracy (Boost Factor ≈ 1.1) while achieving moderate efficiency improvements (Token and Runtime Boost Factors of 1.2 and 1.1). Notably, when compared to CoT with Self-Consistency, our method preserves accuracy (Boost Factor ≈ 1.0) while demonstrating substantial efficiency gains (Token and Runtime Boost Factors of 6.1 and 5.2). This indicates that the benefits of our approach become increasingly pronounced against more complex prompting strategies, particularly in computational efficiency while maintaining accuracy.