International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Design Patterns for Building More Efficient Generative Autonomous Agents: A Survey

Adnan Barwaniwala

Student, The Daly College

Abstract:
This survey paper explores the design patterns essential for constructing generative autonomous agents, a significant advancement over traditional LLM-based applications. These agents can independently plan, execute, and refine tasks, enabling them to overcome limitations such as the need for continuous human input and rules-based reasoning. I review four design patterns: reflection, planning, tool use, and multi-agent collaboration for building more efficient agents. Each is illustrated through a detailed, holistic analysis of a specific implementation technique.

1. Introduction

The emergence of generative AI and large language models (LLMs) like ChatGPT revolutionized the business landscape due to their exceptional capabilities in natural language processing. They quickly found applications in drafting emails, generating content, and even coding, mesmerizing many industries and getting integrated into numerous business workflows.
However, the reliance on human input to prompt and, often, monitor these systems highlighted significant limitations, leading to inefficiencies and scalability issues. This paved the way for the development of generative autonomous agents—advanced AI systems that can independently plan, execute, and refine their own tasks without human oversight. They dynamically interact with their environment, making API calls, monitoring outputs, and using tools to achieve goals.
Generative autonomous agents start by receiving an objective and breaking it down into smaller tasks. They generate specific prompts for each task and iteratively refine their prompts based on the results. This continues until the goal is achieved or regarded as unachievable.
These agents have many profound advantages. Unlike traditional robotic process automation (RPA), which relies on rigid rules-based systems, they offer adaptive, flexible automation that can be applied to a broad range of tasks. For example, in marketing, these agents can autonomously plan and execute campaigns, analyse performance metrics, and enhance strategies based on real-time feedback. Additionally, they can identify target consumers, adjust marketing efforts, and report outcomes, improving efficiency and effectiveness. Moreover, they can revolutionize simulations at scale. By carrying out primary research and creating realistic virtual personas, they can significantly enhance market analysis [1].
These agents' potential extends across various industries, from healthcare, where they assist in diagnostics and treatment planning, to finance, where they upgrade algorithmic trading and fraud detection [2]. Their ability to seamlessly merge with existing systems and adapt to new scenarios without explicitly programming them to do so makes them an extremely integral part of numerous future business operations.

However, these agents have certain inherent limitations. With LLMs as their backbone, high computational costs due to many LLM calls make it very expensive to deploy such agents [3]. Also, such agents may produce factually or logically inaccurate answers along with biased responses due to limited LLM capabilities and biases in LLM training data.

This knowledge of the benefits, potential, and limitations of these agents makes it extremely pertinent that we gain a detailed understanding of efficient design patterns that can be used to build reliable and robust generative autonomous agents. Hence, I present before you a survey of four such design patterns and relevant techniques to implement them: reflection, planning, tool use, and multi-agent collaboration. This paper aims to explain each design pattern with an in-depth analysis of a technique that can be used to implement it, including its overview, performance, benefits, limitations, areas of future research, and some key additional insights. The scope of this paper is limited only to generative autonomous agents and not all autonomous agents. For each design pattern, only one technique implementation, the most commonly used one, has been reviewed. This paper would not have been possible without the newsletter, The Batch, written by Mr. Andrew Ng, where he talks about the importance of efficient design patterns for agentic frameworks in one of his letters and provides relevant literature for each of them.

2. Reflection

Reflection in agentic design involves LLMs iteratively improving their outputs using self-feedback, enhancing performance in tasks like coding, text generation, and question answering [4]. An efficient algorithm to implement this approach in agentic designs is the SELF-REFINE technique [5].

2.1 Overview

The SELF-REFINE algorithm improves LLM outputs through an iterative process involving self-feedback and refinement. Starting with an initial output generated from a task-specific prompt, the model evaluates this output, providing actionable and specific feedback. This feedback guides the refinement of the output, with the process repeating until a stopping condition is met, such as a set number of iterations or achieving satisfactory quality. The model uses few-shot prompting to generate feedback and improvements, making SELF-REFINE highly flexible and efficient, as it does not require supervised training, additional training, or reinforcement learning. Throughout iterations, the model retains the history of feedback and outputs, allowing it to learn from past refinements and avoid repeating mistakes. This approach leverages the model's capability to iteratively enhance its performance across various tasks without needing extensive external data or resources.

2.2 Evaluation

SELF-REFINE was evaluated across various tasks, showcasing significant improvements in output quality. For instance, in code optimization, it increased the optimization rate from 27.3% to 36.0% with GPT-4. In dialogue response generation, the SELF-REFINE preference score jumped from 25.4% to 74.6%. It also showed high gains in sentiment reversal and acronym generation. However, in math reasoning, the improvements were modest due to the nuanced nature of errors. Importantly, GPT-4+SELF-REFINE performs better than GPT-3.5+SELF-REFINE and ChatGPT + SELF-REFINE across all tasks, even in tasks where the initial base results of GPT-4 were lower than GPT-3.5 or ChatGPT. Overall, SELF-REFINE consistently outperformed the base models (GPT-3.5, ChatGPT, and GPT-4) across all tasks.

2.3 Analysis

1. Specific, actionable feedback is crucial in SELF-REFINE as it significantly improved performance in all tasks compared to generic or no feedback.

2. While multiple iterations of SELF-REFINE improve output quality, the gains decrease over time.
3. It struggles with smaller models like Vicuna-13B due to their training on conversational data rather than instructional data.
4. Qualitative analysis of 35 successful and 35 unsuccessful cases reveals that failures often stem from erroneous feedback, with 33% from incorrect error identification and 61% from inappropriate fixes. Conversely, successful cases showed that accurate feedback led to precise improvements in 61% of instances, and even partially correct feedback helped in 33% of cases.
5. Its application in website generation, where it iteratively refines HTML, CSS, and JS from an initial design to enhance usability and aesthetics, demonstrates its potential in real-world, complex tasks.

2.4 Limitations

The main limitation of SELF-REFINE is that it requires base models with strong few-shot learning and instruction-following abilities, which are not always available or open-sourced. Additionally, experiments were conducted using models like GPT-3.5, ChatGPT, GPT-4, and CODEX, which are costly and proprietary, limiting reproducibility. Also, the technique primarily relies on English datasets, reducing its applicability to other languages.

2.5 Additional Insights

The above discussion demonstrates the potential of SELF-REFINE in improving the performance of agentic frameworks, but there are some additional insights I wanted to put forth:

1. Since the publication of the SELF-REFINE technique, advancements in open-source models like the Llama 3 family have made it feasible to integrate SELF-REFINE with these models as well.
2. SELF-REFINE, as a reflection technique, is well-designed but limited by the current capabilities of LLMs. However, with more advanced and intelligent models now available, this technique can be harnessed to create agents that can accurately self-correct and provide factual responses using detailed few-shot prompts and no additional training.
3. Newer models with longer context windows can handle more comprehensive few-shot exemplars, further enhancing performance.
4. Although reflection increases response generation time, it reduces the need for multiple prompts to achieve the desired response, thus balancing quality and user wait time.

3. Planning

Planning in agentic design involves using large language models (LLMs) to independently create and adapt sequences of steps for complex tasks. By breaking down large goals into smaller, more manageable subtasks, LLMs can solve complex challenges dynamically [6]. An efficient way to apply this in agentic designs is via Chain of Thought prompting [7].

3.1 Overview

Chain-of-thought prompting uses few-shot prompting to provide language models with input, chain of thought, and output. The chain of thought represents the intermediate reasoning steps that lead to the final answer. It makes the LLM reasoning process more effective by allowing them to break down complex problems into smaller, manageable steps. It eliminates the need for large training datasets and enables a single model to handle diverse tasks efficiently. It enhances performance in tasks like math word problems, Commonsense reasoning, and symbolic manipulation[1], and the theory, is applicable to any task that humans can solve via language.

3.2 Arithmetic Reasoning

The results and analysis of CoT prompting in arithmetic reasoning demonstrate its effectiveness across multiple benchmarks. It was evaluated using five LLMs[1] on five math word problem datasets[2] and consistently improved performance for models with 100B parameters and above, achieving state-of-the-art results. Ablation studies were performed to confirm the necessity of intermediate reasoning steps in CoT for solving arithmetic problems. CoT prompts were subjected to different annotators, exemplar styles, and orders and still displayed superior performance over standard prompting, demonstrating its robustness.

3.3 Commonsense Reasoning

The common reasoning test of CoT prompting evaluated five datasets: CSQA, StrategyQA, Date Understanding, Sports Understanding, and SayCan. CoT made use of few-shot exemplars and manually composed chains of thought and significantly improved performance. For example, PaLM 540B achieved 75.6% on StrategyQA (up from 69.4%) and 95.4% on Sports Understanding (compared to 84% for a sports enthusiast). These improvements underscore CoT's effectiveness in enhancing reasoning capabilities across diverse commonsense tasks.

3.4 Symbolic Reasoning

The symbolic reasoning test for CoT prompting included last letter concatenation and coin flip tasks, both in-domain and out-of-domain. For in-domain tasks, PaLM 540B achieved nearly 100% accuracy with CoT, while smaller models struggled. Out-of-domain tasks saw improved performance with CoT, but not as high as in-domain results. It demonstrated strong length generalization, especially with larger models like PaLM 540B, which consistently outperformed standard prompting in solving complex symbolic reasoning tasks.

3.5 Limitations and Future Research

CoT faces several limitations. Firstly, while it mimics human reasoning, it's unclear if neural networks are truly "reasoning." Secondly, the manual annotation cost for fine-tuning models with CoT can be prohibitive, but synthetic data generation can help. Thirdly, CoT can lead to incorrect answers due to flawed reasoning paths, highlighting the need for improving factual accuracy. Lastly, CoT's effectiveness with only large model sizes makes it costly for real-world applications, suggesting future research should focus on inducing reasoning in smaller models.

3.6 Additional Insights

1. Since the publication of the paper on CoT, there have been significant advancements in smaller models, with models such as Llama 3 70B (less than 100B parameters) being able to successfully carry out CoT prompting[3].
2. CoT can be combined with SELF-REFINE. Iteratively refining inaccurate chains of thought may allow the LLM to identify reasoning or factual inaccuracies and provide the LLM a chance to improve upon them. This can overcome one of the limitations of CoT mentioned in section 3.5[4].
3. While CoT is suitable for planning, it struggles with complex, multi-step reasoning problems that require planning and tool use. The Chain of Abstraction (CoA) reasoning technique, developed for these scenarios, outperforms CoT prompting, as will be discussed in section 4. However,

[1] GPT-3, LaMDA, PaLM, UL2 20B and Codex.
[2] GSM8K, SVAMP, ASDiv, AQuA, MAWPS.
[3] I confirmed this by asking the Llama 3 70B model to perform CoT on some questions and it successfully did so with accurate results. These results can be viewed in Appendix A.
[4] Appendix A provides evidence for this as well.

understanding CoT is crucial as CoA is derived from it and provides insights into how CoA overcomes CoT's limitations.

4. Tool Use

Tool use in large language models (LLMs) involves integrating external tools like web searches, calculators, or APIs to enhance capabilities. This enables models to access real-time information, perform specialized tasks, and provide more accurate, up-to-date responses [8]. An efficient algorithm to implement this approach in agentic designs is the Chain of Abstraction reasoning method [9].

4.1 Overview

The Chain-of-Abstraction (CoA) reasoning method enhances large language models (LLMs) by separating general reasoning from domain-specific knowledge retrieved from external tools. Initially, LLMs generate abstract reasoning chains using placeholders (e.g., y1, y2). These placeholders are then filled with specific data from external tools like calculators or search engines. This separation allows LLMs to learn holistic reasoning strategies rather than embedding domain-specific knowledge in their parameters. Moreover, it enables parallel processing, where the LLM can generate new reasoning chains while previous ones are being completed with data retrieved from tools, significantly improving inference efficiency.

4.2 Fine-Tuning Data Generation

LLMs were fine-tuned to perform CoA reasoning. To generate data for fine-tuning LLMs with CoA reasoning, researchers collect question-answer samples from existing open-source QA datasets. Using LLaMa-70B, they prompt it to rewrite answers by labeling spans corresponding to knowledge operations, such as math derivations or Wikipedia references. These labeled spans are replaced with abstract placeholders to create fillable CoA traces. These traces are checked using specialized tools, retaining only those questions whose CoA leads to the correct results.

4.3 EXPERIMENTAL SETTINGS

CoA is tested on mathematical reasoning using GSM8K and ASDiv datasets for in-distribution evaluation, and SVAMP and MAWPS for zero-shot evaluation, with an equation solver offered as a tool. For Wikipedia QA, CoA trains LLMs to identify and reason with Wikipedia articles and entities using HotpotQA, and is evaluated on WebQuestions, NaturalQuestions, and TriviaQA, using a BM25 Wikipedia search engine and the SpaCy library for named entity recognition. Various LLaMa models were fine-tuned and compared against baselines like few-shot prompting, fine-tuning with CoT data, Toolformer, and more for evaluation purposes.

4.4 Results and Analysis

Mathematical Reasoning:

1. CoA outperforms baselines like CoT-FSP, CoT-FT[5], and Toolformer, demonstrating robust and efficient reasoning in both in-distribution and out-of-distribution evaluations with LLaMa models.
2. The CoA (no Tool) ablation study[6] shows it exceeds all baselines on zero-shot generalization, highlighting the robustness of abstract reasoning chains.
3. CoA excels in tasks requiring lengthy reasoning chains, producing reasoning steps closely aligned with the true reasoning and outperforming CoT-FSP and CoT-FT, especially in multi-step reasoning problems.

---

[5] CoT-FSP is Chain of Thought Few-Shot Prompting and CoT-FT is Chain of Thought Fine Tuning.

[6] The CoA (no tool) ablation study involves testing CoA without using any external tools to evaluate the impact of tool usage on model performance.

4. A human evaluation on GSM8K confirms CoA eliminates arithmetic errors and significantly reduces reasoning errors compared to baselines.
5. It excels in multi-step reasoning by separating reasoning from tool use, enabling parallel tool calls, and reducing API delays, leading to improved scalability and lower inference times compared to Toolformer.

Wiki QA:

The CoA method, tested on Wiki QA tasks using LLaMa-2-Chat models, outperforms baselines like CoT-FSP, CoT-FT, Toolformer, and FireAct[7] by achieving higher exact match rates, better performance on challenging bridge-type questions and faster inference times. It also excels in zero-shot generalization on datasets like NaturalQuestions, TriviaQA[8] and WebQuestions[9].

4.5 Limitations and Future Research

The CoA method's limitations include the use of incomplete datasets, reliance on English, and employing greedy decoding, while more advanced strategies like self-consistency decoding could have been applied. Initial integration of CoA with self-consistency showed promise, suggesting further research in the area. It also requires significant computational resources for fine-tuning full LLMs, highlighting the need for exploring more efficient training methods like LoRA in future work.

4.6 ADDITIONAL INSIGHTS

1. Tools greatly enhance agent capabilities, improving functionality and autonomy. Techniques like Chain-of-Thought prompting with tool use, ReAct, and Chain of Abstraction (CoA) reasoning have been developed to optimize tool efficiency. CoA mostly excels amongst these methods, but further research is still needed. Efficient training methods, as mentioned previously, are one area of research, and another very promising area is CoA + few-shot prompting, especially with the increased context windows of LLMs like Google's Gemini 1.5 Pro (1 million tokens) and advanced instruction following in models like OpenAI's GPT-4o.
2. Multi-step reasoning that requires tool use involves planning and tool execution. While CoT prompting is effective for planning, it often causes issues like incorrect reasoning chains and longer inference times when tools are involved. CoA reasoning addresses these problems, leading to better planning and tool execution in such tasks.
3. For tool use, an LLM must return a structured output with the function name and the arguments. This is achieved through few-shot prompting or fine-tuning LLMs for function-calling. Smaller models often hallucinate or generate incorrect structured outputs, but reflection may help them iteratively improve. However, this may significantly increase response times. Hence, this is also a very viable field for future research.
4. Self-refine, as mentioned in section 2.2, struggled with math reasoning. However, by combining self-refine with tool use, its true potential can be unlocked. This can be done in two ways: use an external tool that verifies the correctness of LLM-generated answers or have the LLM generate code that is executed to reach the answer. This can result in the LLM more accurately identifying errors and appropriate fixes while providing feedback.

[7] FireAct fine-tunes LLMs on HotpotQA using ReAct trajectories distilled from GPT-4 to improve performance on multi-step reasoning tasks.
[8] CoA outperforms the baselines for both the sets
[9] CoA matches the best possible baselines in this dataset.

5. MULTI-AGENT COLLABORATION

Multi-agent collaboration in AI involves breaking down complex tasks into subtasks managed by different agents, each playing a specific role. This approach optimizes task execution and enhances performance by leveraging the unique capabilities of individual agents [11]. An open-source library to efficiently implement this in agentic designs is Autogen [12].

5.1 Overview

AutoGen is a multi-agent conversation framework designed to simplify the creation of complex LLM applications. It includes customizable agents that can use LLMs, human inputs, and tools for diverse roles like code writing and validation. These agents handle multi-turn interactions by themselves through unified conversation interfaces and auto-reply mechanisms. The framework focuses on a conversation programming paradigm based on computation and conversation-driven control flow. Computation involves the actions agents take to process responses, while control flow dictates the sequence and conditions of these computations based on the conversation context. This allows for dynamic task management and flexible, role-based agentic group interactions.

5.2 Performance

AutoGen is used in different applications to illustrate its innovative potential, problem-solving capabilities, and ability to simplify the development of complex LLM applications:

1. **Math Problem-Solving:**

Built-in AutoGen agents were reused to create a multi-agent system for math reasoning that outperformed alternatives like Multi-Agent Debate, LangChain ReAct, vanilla GPT-4, ChatGPT + Code Interpreter, and ChatGPT + Plugin (Wolfram Alpha) on the MATH dataset, highlighting the superior processing abilities of multi-agent systems over single-agent systems. Additionally, being able to involve human input while problem-solving allowed the multi-agent system to solve many complex, high-level problems as well.

2. **Decision-Making in Text-Based Environments:**

A multi-agent system was tested on ALFWorld tasks. A planning agent decided the plan of steps to take, a grounding agent provided common-sense knowledge about the physical world in case the plan lacked it, and an execution agent performed the tasks in the ALFWorld environments. This role-based, multi-agent design allowed it to excel at the tasks.

3. **Dynamic Group Chat:**

AutoGen agents can share context and converse dynamically, with a GroupChatManager dynamically selecting the next agent to speak, collecting its response, and broadcasting it to all other agents. This allows agents to be very flexible and deal with unforeseen circumstances extremely efficiently compared to rule-based agent conversations.

4. **Conversational Chess:**

AutoGen was used to create an application to play chess via a natural language interface. Customizable agents that could either be human or LLM players and a board agent that checks for valid moves were used. It offers different modes: human vs human, human vs AI, and AI vs AI (modes can be switched during the game as well). The board agent can be adjusted in response to evolving game rules or varying chess rule variants. Overall, this application serves to highlight the immense innovative potential of multi-agent systems.

5.3 Limitations

AutoGen has several limitations: it requires robust privacy measures to protect user data during human-agent interactions; it may exhibit biases inherent in LLM training data, requiring bias mitigation in agent-

agent and human-agent conversations for fairness; and it poses accountability challenges due to its multi-agent structure, requiring clear mechanisms to trace the decision-making processes between agents.

5.4 Additional Insights

1. Multi-agent systems enhance agentic frameworks by utilizing specialized agents instead of a single, broadly capable agent. AutoGen evaluations showed that multi-agent systems outperform single agents, as LLMs excel with specialized tasks and detailed instructions. Frameworks like AutoGen, CrewAI, and LangGraph further improve multi-agent systems by enabling multi-agent tool use and flexible, dynamic inter-agent conversations.
2. Reflection can be enhanced by adapting the SELF-REFINE technique within a multi-agent framework. One agent can generate outputs, while another provides constructive criticism, improving responses through their interaction. This method can also benefit from human input guiding the self-reflection process, potentially surpassing the standard algorithm.
3. Multi-agent systems can be integrated with planning by having an LLM create a task plan and allocate different aspects of the plan to specialized agents which the LLM creates itself by generating the required code. Tool use can execute the code for setting up the agents, while reflection can improve the planning process, agent creation, and responses of individual agents. A single agent can handle planning and tool use within the system.

6. CONCLUSION

In a world where generative autonomous agents are the future, this paper presented 4 efficient design patterns for building these agents, leveraging their full potential and overcoming their inherent limitations. We started with reflection and understood the potential of SELF-REFINE. We then explored planning and understood Chain of Thought prompting which was followed by tool use and Chain of Abstraction reasoning that built on CoT and lastly, we dove into the massive potential of multi-agent collaboration which can be implemented via AutoGen. We gained a holistic understanding of each technique by reviewing its performance, benefits and limitations. Many viable areas of future research were suggested and key insights into how these techniques can work together and how they connect to recent advancements in AI were provided. To conclude, after reading this paper I hope you've gained a keen understanding into creating more robust and streamlined agents.

A. Chain of Thought Prompting Results with Llama 3 70 B model

To start off, I tested the model for CoT prompting on the 1st question of SET 1 adapted from the following pdf: https://www.ctanujit.org/uploads/2/5/3/9/25393293/bs_book.pdf. The question was: "How many zeros are at the end of 1000!?"

Step 4: Count the factors of 5

To count the factors of 5, let's divide 1000 by 5 and its powers:

* 1000 ÷ 5 = 200 (200 numbers contribute one factor of 5 each)

* 200 ÷ 5 = 40 (40 numbers contribute an additional factor of 5 each)

* 40 ÷ 5 = 8 (8 numbers contribute yet another factor of 5 each)

* 8 ÷ 5 = 1 (1 number contributes one more factor of 5)

Adding up the factors of 5, we get: 200 + 40 + 8 + 1 = 249

Step 5: Determine the number of zeros

Since we have 249 factors of 5, and each factor of 10 requires one factor of 5 and one factor of 2, we know that there are at least 249 zeros at the end of 1000!.

However, we need to check if there are enough factors of 2 to pair with these factors of 5. Since there are many more factors of 2 than 5 in 1000!, we can be certain that there are enough factors of 2 to pair with the 249 factors of 5.

Step 6: Conclusion

Therefore, there are exactly 249 zeros at the end of 1000!.

Figure 1: Chain-of-Thought Successfully Generated by the Llama Model.

Figure 1 shows that the model was accurately able to perform CoT on the question, even without the need for few-shot exemplars, and was able to reach the correct answer as a result.

Interestingly, I tested the model on another question for further validation. When I simply asked it to solve the question using chain-of-thought reasoning (CoT), it was able to generate a chain of thought, but that led to an incorrect answer. However, when I asked the model to generate a chain of thought and reflect on it, the model was able to identify its error and provide a correct answer as well. The question was: "There are three bags, each containing two marbles. Bag A contains two white marbles, Bag B contains two black marbles, and Bag C contains one white marble and one black marble. You pick a random bag and take out one marble, which is white. What is the probability that the remaining marble from the same bag is also white?"

This was the 19th question adapted from the following website: https://parade.com/970343/parade/logic-puzzles/.

Using the law of total probability, we get:

P(White Remaining) = P(Bag A | White Marble) * P(White Remaining | Bag A) + P(Bag C | White Marble) * P(White Remaining | Bag C)

= (1/2) * 1 + (1/2) * (1/2)

= 3/4

So, the probability that the remaining marble from the same bag is also white is 3/4 or 75%.

Figure 2: Last Part of the Llama 3 Chain-of-Thought with the Incorrect Answer Generated.

Figure 2 shows that the model generated an incorrect answer when only asked to perform CoT on the second question.

= 3/4

Critical Analysis:

Let's review our reasoning:

* We correctly eliminated Bag B since it only has black marbles.

* We correctly assigned equal probabilities to Bag A and Bag C.

* We correctly calculated the probability of the remaining marble being white for each bag.

* However, our combination of probabilities might be incorrect. We need to reconsider how to combine these probabilities.

Revised Reasoning:

Let's use Bayes' Theorem to update our probabilities:

P(Bag A | White Marble) = P(White Marble | Bag A) * P(Bag A) / P(White Marble)

= 1 * (1/3) / (1 * (1/3) + 0 * (1/3) + (1/2) * (1/3))

= 2/3

P(Bag C | White Marble) = P(White Marble | Bag C) * P(Bag C) / P(White Marble)

= (1/2) * (1/3) / (1 * (1/3) + 0 * (1/3) + (1/2) * (1/3))

= 1/3

Now, we can calculate the probability of the remaining marble being white:

P(White Remaining) = P(Bag A | White Marble) * P(White Remaining | Bag A) + P(Bag C | White Marble) * P(White Remaining | Bag C)

= (2/3) * 1 + (1/3) * (1/2)

= 2/3

Final Answer:

The probability that the remaining marble from the same bag is also white is 2/3 or approximately 66.67%.

Figure 3: Portion of the Model Response When it was Asked to Perform CoT and Reflection.

Figure 3 shows that the model generated an incorrect answer at first, but upon reflection, it identified its error and corrected its chain of thought to reach the correct answer.

This showcases the advancements in smaller LLMs and their potential to be part of generative autonomous agents built with the design patterns discussed. Furthermore, it demonstrates how planning can be successfully combined with self-reflection to produce more efficient results, as mentioned previously.

References

1. Mikhail B., François C., Gaurav J., Daniel S., Leonid Z., David M., “GPT Was Just the Beginning. Here Come Autonomous Agents”. https://www.bcg.com/publications/2023/gpt-was-only-the-beginning-autonomous-agents-are-coming/
2. Jagreet G. “Generative AI Agents”. https://www.xenonstack.com/blog/generative-ai-agents/
4. Tanya M. “Breaking Down AutoGPT: What It Is, Its Features, Limitations, Artificial General Intelligence (AGI) And Impact of Autonomous Agents on Generative AI”. https://www.marktechpost.com/2023/07/11/breaking-down-autogpt-what-it-is-its-features-limitations-artificial-general-intelligence-agi-and-impact-of-autonomous-agents-on-generative-ai/

5. Andrew N. “Agentic Design Patterns Part 2, Reflection”. https://www.deeplearning.ai/the-batch/agentic-design-patterns-part-2-reflection/
6. Aman M., Niket T., Prakhar G., Skyler H., Luyu G., Sarah W., Uri A., Nouha D., Shrimai P., Yiming Y., Shashank G., Bodhisattwa P. M., Katherine H., Sean W., Amir Y., Peter C., “Self-Refine: Iterative Refinement with Self-Feedback”, arXiv preprint, 2023, 1-10.
7. Andrew N. “Agentic Design Patterns Part 4, Planning”. https://www.deeplearning.ai/the-batch/agentic-design-patterns-part-4-planning/
8. Jason W., Xuezhi W., Dale S., Maarten B., Brian I., Fei X., Ed C., Quoc L., Denny Z., “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”, arXiv preprint, 2022, 1-9
9. Andrew N. “Agentic Design Patterns Part 3, Tool Use”. https://www.deeplearning.ai/the-batch/agentic-design-patterns-part-3-tool-use/
10. Silin G., Jane D., Ping Y., Xiaoqing T., Ramakanth P., Olga G., Koustuv S., Asli C., Antoine B., Tianlu W., “Efficient Tool Use with Chain-of-Abstraction Reasoning”, arXiv preprint, 2024, 1-9.
1. 10 Sundar P., Demis H., “Our next-generation model: Gemini 1.5”, https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/
11. Andrew N. “Agentic Design Patterns Part 5, Multi-Agent Collaboration”. https://www.deeplearning.ai/the-batch/agentic-design-patterns-part-5-multi-agent-collaboration/
12. Qingyun W., Gagan B., Jieyu Z., Yiran W., Beibin L., Erkang Z., Li J., Xiaoyun Z., Shaokun Z., Jiale L., Ahmed A., Ryen W., Doug B., Chi W., “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation”, arXiv preprint, 2023, 1-10.