

GÖDEL AGENT: A SELF-REFERENTIAL FRAMEWORK FOR AGENTS RECURSIVELY SELF-IMPROVEMENT

Anonymous authors

Paper under double-blind review

ABSTRACT

The rapid advancement of large language models (LLMs) has significantly enhanced the capabilities of AI-driven agents across various tasks. However, existing agentic systems, whether based on fixed pipeline algorithms or pre-defined meta-learning frameworks, cannot search the whole agent design space due to the restriction of human-designed components, and thus might miss the globally optimal agent design. In this paper, we introduce Gödel Agent, a self-evolving framework inspired by the Gödel machine, enabling agents to recursively improve themselves without relying on predefined routines or fixed optimization algorithms. Gödel Agent leverages LLMs to dynamically modify its own logic and behavior, guided solely by high-level objectives through prompting. Experimental results on multiple domains including coding, science, and math demonstrate that implementation of Gödel Agent can achieve continuous self-improvement, surpassing manually crafted agents in performance, efficiency, and generalizability.

1 INTRODUCTION

As large language models (LLMs) such as GPT-4 (OpenAI et al., 2024) and LLaMA3 (Dubey et al., 2024) demonstrate increasingly strong reasoning and planning capabilities, LLM-driven agentic systems have achieved remarkable performance in a wide range of tasks (Wang et al., 2024a). Substantial effort has been invested in manually designing sophisticated agentic systems using human priors in different application areas. Recently, there has been a significant interest in creating self-evolving agents with minimal human effort, which not only greatly reduces human labor but also produces better solutions by incorporating environmental feedback. Given that human effort can only cover a small search space of agent design, it is reasonable to expect that a self-evolving agent with the freedom to explore the full design space has the potential to produce the global optimal solution.

There is a large body of work proposing agents capable of self-refinement. However, there are inevitably some human priors involved in these agent designs. Some agents are designed to iterate over a fixed routine consisting of a list of fixed modules, while some of the modules are capable of taking self- or environment feedback to refine their actions (Shinn et al., 2024; Chen et al., 2023b; Qu et al., 2024; Yao et al., 2023). This type of agent, referred to as **Hand-Designed Agent**, is depicted as having the lowest degree of freedom in Figure 1. More automated agents have been designed to be able to update their routines or modules in some pre-defined meta-learning routine, for example, natural language gradients (Zhou et al., 2024), meta agent (Hu et al., 2024), or creating and collecting demonstrations (Khatab et al., 2023). This type of agent, known as **Meta-Learning Optimized Agents**, is depicted as having the middle degree of freedom in Figure 1.

It is evident that both types of agents above are inherently constrained by human priors and one intuitional method to further increase the freedom of self-improvement is to design a meta-meta-learning algorithm, to learn the meta-learning algorithm. However, there is always a higher-level meta-learning algorithm that can be manually designed to learn the current-level meta-learning method, creating a never-ending hierarchy of meta-learning.

In this paper, we propose **Gödel Agent** to eliminate the human design prior, which is an automated LLM agent that can freely decide its own routine, modules, and even the way to update them. It is inspired by the self-referential Gödel machine (Schmidhuber, 2003), which was originally proposed to solve formal proof problems and was proven to be able to find the global optimal solutions. *Self-reference* means the property of a system that can analyze and modify its own code, including the

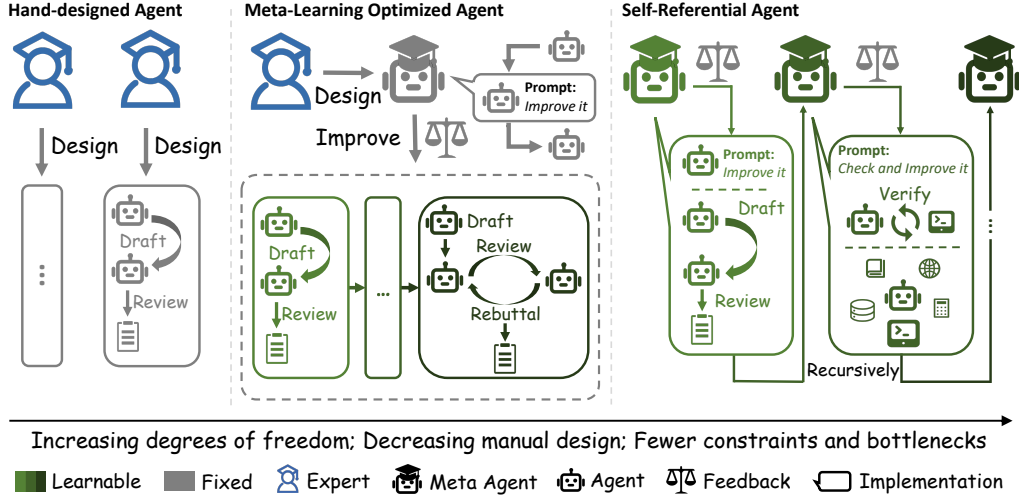


Figure 1: Comparison of three agent paradigms. Hand-designed agents rely on human expertise which are limited in scope and labor-intensive. Meta-learning optimized agents are constrained by a fixed meta-learning algorithm, restricting their search space and optimization potential. In contrast, self-referential agent (Gödel Agent) can recursively improve itself without any limitation. Note that the input to Gödel Agent is itself, allowing it to modify itself and output a new version of itself.

parts responsible for the analysis and modification processes (Astrachan, 1994). Therefore, it can achieve what’s known as “*recursive self-improvement*”, where it iteratively updates itself to become more efficient and effective at achieving its predefined goals. In this case, Gödel Agent can analyze and modify its own code, including the code for analyzing and modifying itself, and thus can search the full agent design space, which is depicted as having the highest degree of freedom in Figure 1. Gödel Agent can theoretically make increasingly better modifications over time through recursively self-update (Yampolskiy, 2015; Wang, 2018).

In this paper, we choose to implement it by letting it manipulate its own runtime memory, i.e., the agent is able to retrieve its current code in the runtime memory and modify it by *monkey patching*, which dynamically modifies classes or modules during execution. In our implementation, we adhere to a minimalist design to minimize the influence of human priors. We implement the optimization module using a recursive function. In this module, LLM analyzes and makes a series of decisions, including reading and modifying its own code from runtime memory (*self-awareness* and *self-modification*), executing Python or Linux commands, and interacting with the environment to gather feedback. The agent then proceeds to the subsequent recursive depth and continues to optimize itself. It is worth noting that the optimization module may have already been modified by the time the recursion occurs, potentially enhancing its optimization capabilities.

To validate the effectiveness of Gödel Agent, we conduct experiments on multiple domains including coding, science, math, and reasoning. Our experimental results demonstrate that Gödel Agent achieves significant performance gain across various tasks, surpassing various widely-used agents that require human design. The same implementation of Gödel Agent can easily adapt to different tasks by only specifying the environment description and feedback mechanism. Additionally, the case study of the optimization progress reveals that Gödel Agent can provide novel insights into agent design. We also investigate the impact of the initial policy for improvement on subsequent outcomes, finding that a good start can significantly accelerate convergence during optimization.

In summary, our contributions are as follows:

- We propose the first self-referential agent framework, Gödel Agent, based on LLMs. It autonomously engages in self-awareness, self-modification, and recursive self-improvement across any task, reducing the need for manual agent design and offering higher flexibility and freedom.
- We implement Gödel Agent framework using the monkey patching method. Our experiments show that Gödel Agent outperforms manually designed agents and surpasses its earlier versions on several foundational tasks, demonstrating effective self-improvement.

- We analyze Gödel Agent’s optimization process, including its self-referential capabilities and the resulting agentic system, aiming to deepen our understanding of both LLMs and agentic systems.
- Our framework offers a promising direction for developing flexible and capable agents through recursive self-improvement.

2 METHOD

In this section, we first describe the formal definitions for previous agent methods with a lower degree of freedom, including hand-design and meta-learning optimized agents, as a background. Then we introduce our proposed Gödel Agent, a self-referential agent that can recursively update its own code, evolving over training.

Let $\mathcal{E} \in \mathcal{S}$ denote a specific environment state, where \mathcal{S} denotes the set of all possible environments the agent will encounter. For example, an environment can be a mathematical problem with ground truth solutions. We denote the policy that an agent follows to solve a problem in the current environment by $\pi \in \Pi$, where Π is the set of all possible policies the agent can follow.

A **hand-designed agent**, as shown in the left panel of Figure 1, is not capable of updating its policy and following the same policy π all the time, regardless of environmental feedback.

In contrast, a **meta-learning optimized agent** updates its policy based on a meta-learning algorithm I at training time based on the feedback it receives from the environment, as shown in the middle panel of Figure 1. The environment feedback is usually defined as a utility function $U : \mathcal{S} \times \Pi \rightarrow \mathbb{R}$, which maps an environment and a policy to a real-valued performance score. The main training algorithm of a meta-learning optimized agent can then be written as follows:

$$\pi_{t+1} = I(\pi_t, r_t), \quad r_t = U(\mathcal{E}, \pi_t),$$

In this case, the agent’s policy π_t evolves at training time, with the learning algorithm I updating the policy based on feedback r_t , while the meta-learning algorithm I remains fixed all the time.

A **self-referential Gödel Agent**, on the other hand, updates both the policy π and the meta-learning algorithm I recursively. The main idea is that, after each update, the whole code base of the agent is rewritten to accommodate any possible changes. Here we call this self-updatable meta-learning algorithm I a self-referential learning algorithm. The training process of a Gödel Agent can then be written as:

$$\pi_{t+1}, I_{t+1} = I_t(\pi_t, I_t, r_t, g), \quad r_t = U(\mathcal{E}, \pi_t),$$

where $g \in \mathcal{G}$ represents the high-level goal of optimization, for example, solving the given mathematical problem with the highest accuracy. Such a recursive design of the agent requires the specification of an initial agent algorithm (π_0, I_0) , detailed as follows:

- A initial agent policy π_0 to perform the desired task within the environment \mathcal{E} . For example, it can be chain-of-thought prompting of an LLM.
- A self-referential learning algorithm I_0 for recursively querying an LLM to rewrite its own code based on the environmental feedback.

We then further specify a possible initialization of the self-referential learning algorithm $I_0 = (f_0, o_0)$, using a mutual recursion between a decision-making function f_0 , and an action function o_0 :

- The decision-making function f_0 , implemented by an LLM, determines a sequence of appropriate actions $a_1, a_2, \dots, a_n \in \mathcal{A}$ based on the current environment \mathcal{E} , the agent’s algorithm (π_t, I_t) , and the goal g .
- The action function o_0 , executes the selected action and updates the agent’s policy accordingly.

The set of actions \mathcal{A} for the action function o to execute needs to include the following four actions:

- `self_inspect`: Introspect and read the agent’s current algorithm (π_t, I_t) .
- `interact`: Interact with the environment by calling the utility function U to assess the performance of the current policy π_t .

Algorithm 1 Recursive Self-Improvement of Gödel Agent

```

1: Input: Initial agent policy  $\pi_0$ , initial decision function  $f_0$ , goal  $g$ , environment state  $\mathcal{E}$ , utility function  $U$ , self code reading function SELF_INSPECT
2: Output: Optimized policy  $\pi$  and Gödel Agent  $s$ 
3:  $\triangleright$  Get all agent code, including the code in this algorithm.
4:  $s \leftarrow \text{SELF\_INSPECT}()$ 
5:  $\triangleright$  Compute the initial performance.
6:  $r \leftarrow U(\mathcal{E}, \pi_0)$ 
7:  $\triangleright$  Perform recursive self-improvement.
8:  $\pi, s \leftarrow \text{SELF\_IMPROVE}(\pi, s, r, g)$ 
9: return  $\pi, s$ 
10:
11:  $\triangleright$  Initial code of self-referential learning.
12: function SELF_IMPROVE( $\mathcal{E}, \pi, s, r, g$ )
13:    $\triangleright$  Obtain action sequence.
14:    $a_1, \dots, a_n \leftarrow f_0(\pi, s, r, g)$ 
15:   for  $a_i$  in  $a_1, \dots, a_n$  do
16:      $\pi, s, r \leftarrow \text{EXECUTE}(\mathcal{E}, \pi, s, r, a_i)$ 
17:   end for
18:   return  $\pi, s$ 
19: end function
20:
21:  $\triangleright$  Initial action execution function.
22: function EXECUTE( $\mathcal{E}, \pi, s, r, a$ )
23:   switch  $a.name$ 
24:     case self.state:
25:        $s \leftarrow \text{SELF\_INSPECT}()$ 
26:     case interact:
27:        $r \leftarrow U(\mathcal{E}, \pi)$ 
28:     case self.update:
29:        $\pi, s \leftarrow a.code$ 
30:     case continue_improve:
31:        $\triangleright$  Recursively invoke self-improvement.
32:        $\pi, s \leftarrow \text{SELF\_IMPROVE}(\mathcal{E}, \pi, s, r, g)$ 
33:   return  $\pi, s, r$ 
34: end function

```

- `self.update`: Alter and update (π_t, I_t) with an LLM and produce (π_{t+1}, I_{t+1}) .
- `continue_improve`: If no other actions can be taken, recursively invoke the decision algorithm f to produce new actions.

The agent code is updated to (π_{t+1}, I_{t+1}) after the current execution of (π_t, I_t) is finished. Both the agent algorithm (π, I) and the action set \mathcal{A} are not static and can be expanded and modified by the agent itself at the training time. Algorithm 1 illustrates the described algorithm for the Gödel Agent. Each recursive call enables the agent to refine its performance and become progressively more efficient.

3 GÖDEL AGENT INITIALIZATION

There are various ways to initiate a Gödel Agent. Any specific agent instance during the recursive optimization process can be viewed as an instantiation of the Gödel Agent. Our implementation leverages runtime memory interaction techniques to enable self-awareness and self-modification, as illustrated in Figure 2. These techniques include dynamic memory reading and writing (*monkey patching*) to facilitate recursive self-improvement. Additionally, we have incorporated several auxiliary tools to accelerate the convergence of the Gödel Agent’s optimization process.

3.1 IMPLEMENTATION DETAILS

The core functionalities of our Gödel Agent are outlined below:

Self-Awareness via Runtime Memory Inspection Our Gödel Agent achieves self-awareness by inspecting runtime memory, particularly local and global variables in Python. This capability allows the agent to extract and interpret the variables, functions, and classes that constitute both the environment and the agent itself, according to the modular structure of the system. By introspecting these elements, the agent gains an understanding of its own operational state and can adapt accordingly.

Self-Improvement via Dynamic Code Modification Gödel Agent can engage in reasoning and planning to determine whether it should modify its own logic. If modification is deemed necessary, Gödel Agent generates new code, dynamically writes it into the runtime memory, and integrates it into its operational logic. This dynamic modification allows it to evolve by adding, replacing, or removing logic components as it encounters new challenges, thus achieving self-improvement.

Environmental Interaction To assess performance and gather feedback, Gödel Agent is equipped with interfaces for interacting with its environment. Each task provides tailored envi-

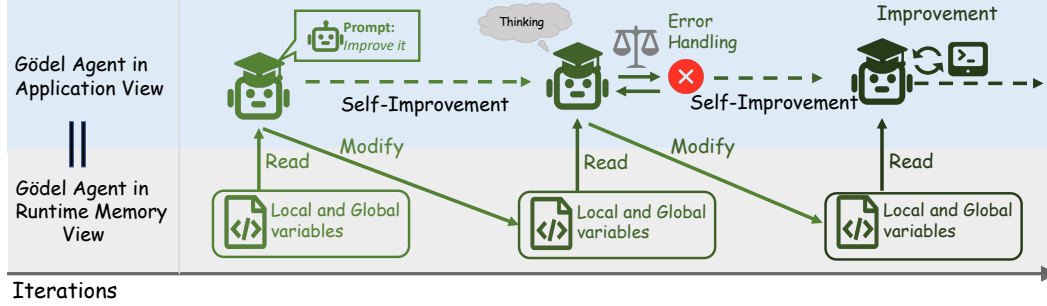


Figure 2: An illustration of our implementation of Gödel Agent. It employs monkey patching to directly read and modify its own code in runtime memory, enabling self-awareness and self-modification.

ronmental interfaces, enabling it to evaluate its performance and adjust its strategies accordingly. This interaction is a crucial part of the feedback loop in the recursive improvement process.

Recursive Improvement Mechanism At each time step, Gödel Agent determines the sequence of operations to execute, which includes reasoning, decision-making, and action execution. After completing the operations, Gödel Agent evaluates whether its logic has improved and decides whether to proceed to the next recursive iteration. Over successive iterations, Gödel Agent’s logic evolves, with each step potentially improving its decision-making capacity.

Goal Prompt and Task Handling The goal prompt informs Gödel Agent that it possesses the necessary privileges to enhance its logic and introduces the available tools for improvement. As shown in Appendix A, this prompt encourages Gödel Agent to fully explore its potential and leverage the tools for self-optimization. To ensure effectiveness across diverse tasks, we provide Gödel Agent with an initial policy, where it will start to explore different policies to analyze its efficiency in optimizing performance.

3.2 ADDITIONAL DESIGNS TO SUPPORT GÖDEL AGENT’S OPTIMIZATION

While the core functionality of Gödel Agent theoretically allows limitless self-improvement, current LLMs exhibit limitations. To address these challenges, we have integrated several supportive mechanisms to enhance Gödel Agent’s performance:

Thinking Before Acting Gödel Agent is capable of deferring actions to first reason about the situation, allowing it to output reasoning paths and analysis without immediately executing any operations. This approach enhances the quality of decision-making by prioritizing planning over hasty action.

Error Handling Mechanism Errors during execution can lead to unexpected terminations of the agent process. To mitigate this, we implement a robust error recovery mechanism. If an operation results in an error, Gödel Agent halts the current sequence and moves on to the next time step, carrying forward the error information to improve future decisions.

Additional Tools We also equipped Gödel Agent with additional potentially useful tools, such as the ability to execute Python or Bash code and call LLM API.

Although these additional tools are not strictly necessary for self-improvement, their inclusion accelerates the convergence of Gödel Agent’s recursive optimization process. We conducted ablation studies to assess the effectiveness of these tools, as discussed in Section 5.1.

4 EXPERIMENTS

We conduct a series of experiments across multiple tasks, including reading comprehension, mathematics, reasoning, and multitasking. These experiments are designed to evaluate Gödel Agent’s self-improvement capabilities in comparison to both hand-designed agents and a state-of-the-art au-

tomated agent design method. In addition, to gain deeper insights into the behavior and performance of Gödel Agent, we also conduct a case study with Game of 24 as presented in Section 5.3.

4.1 BASELINE METHODS

To establish a comprehensive baseline, we select both fixed hand-designed methods and a representative automated agent design technique. Our hand-designed methods are well-known approaches that focus on enhancing reasoning and problem-solving capabilities. These include: 1) Chain-of-Thought (CoT) (Wei et al., 2022) that encourages agents to articulate their reasoning processes step-by-step before providing an answer. 2) Self-Consistency with Chain-of-Thought (CoT-SC) (Wang et al., 2023b) that generates multiple solution paths using the CoT framework and selects the most consistent answer. 3) Self-Refine (Madaan et al., 2024) that involves agents assessing their own outputs and correcting mistakes in subsequent attempts. 4) LLM-Debate (Du et al., 2023) that allows different LLMs to engage in a debate, offering diverse viewpoints. 5) Step-back Abstraction (Zheng et al., 2024) that prompts agents to initially focus on fundamental principles before diving into task details. 6) Quality-Diversity (QD) (Lu et al., 2024) that generates diverse solutions and combines them. 7) Role Assignment (Xu et al., 2023) that assigns specific roles to LLMs to enhance their ability to generate better solutions by leveraging different perspectives. Given the limitations of fixed algorithms in handling dynamic scenarios, we select 8) Meta Agent Search (Hu et al., 2024), the latest state-of-the-art method for automated agent design, as our main comparison point.

4.2 EXPERIMENTAL SETTINGS

Following the setup of Hu et al. (2024), we evaluate Gödel Agent’s self-improvement capabilities across four well-known benchmarks. The benchmarks are as follows: 1) DROP (Dua et al., 2019) for reading comprehension. 2) MGSM (Shi et al., 2022) for testing mathematical skills in a multilingual context. 3) MMLU (Hendrycks et al., 2021) for evaluating multi-task problem-solving abilities. 4) GPQA (Rein et al., 2023) for tackling challenging graduate-level science questions.

Given the complexity of the tasks and the need for advanced reasoning and understanding, the improvement cycle of Gödel Agent is driven by GPT-4o. In the main experiment, we implement two different settings: 1) To make a fair comparison with baseline methods, we forbid Gödel Agent to change the API of the LLM used to perform the tasks (by default GPT-3.5) and use a closed-book approach with no access to the Internet, and 2) To explore the upper bound of Gödel Agent’s capabilities, we remove all constraints. Chain of Thought is applied as the initial policy for all tasks, given its simplicity and versatility. In addition, as shown in Section 5.3, we also analyze the performance of Gödel Agent when using other algorithms as the initial policies.

We perform 6 independent self-improvement cycles for each task. Each cycle represents a complete self-improvement process, where Gödel Agent iteratively modifies its logic to enhance performance. Further details regarding the experimental setup and additional results can be found in Appendix B.

4.3 EXPERIMENTAL RESULTS AND ANALYSIS

The experimental results on the four datasets are shown in Table 1. Under the same experimental settings, Gödel Agent achieves either optimal or comparable results to Meta Agent Search across all tasks. Notably, in the mathematics task MGSM, Gödel Agent outperforms the baseline by 11%. This suggests that reasoning tasks offer greater room for improvement for Gödel Agent, while in the knowledge-based QA dataset, it only slightly surpasses baselines. In contrast to Meta Agent Search, which relies on manually designed algorithmic modules to search, Gödel Agent demonstrates greater flexibility. It requires only a simple initial policy, such as CoT, with all other components being autonomously generated. Moreover, through interaction with the environment, Gödel Agent gradually adapts and independently devises effective methods for the current task. The final policies generated by Gödel Agent for four tasks are shown in Appendix C.1. Additionally, our method converges faster, with the required number of iterations and computational cost across different tasks compared to the Meta Agent shown in Appendix D.

We also conduct experiments without restrictions, where Gödel Agent significantly outperforms all baselines. Upon further analysis, we find that this is primarily due to the agent’s spontaneous requests for assistance from more powerful models such as GPT-4o in some tasks. Therefore, Gödel

Table 1: Results of three paradigms of agents on different tasks. The highest value is highlighted in **bold**, and the second-highest value is underlined. Gödel-base is the constrained version of Gödel Agent, allowing for fair comparisons with other baselines. Gödel-free represents the standard implementation without any constraints, whose results are *italicized*. We report the test accuracy and the 95% bootstrap confidence interval on test sets².

Agent Name	F1 Score		Accuracy (%)	
	DROP	MGSM	MMLU	GPQA
Hand-Designed Agent Systems				
Chain-of-Thought (Wei et al., 2022)	64.2 \pm 0.9	28.0 \pm 3.1	65.4 \pm 3.3	29.2 \pm 3.1
COT-SC (Wang et al., 2023b)	64.4 \pm 0.8	28.2 \pm 3.1	65.9 \pm 3.2	30.5 \pm 3.2
Self-Refine (Madaan et al., 2024)	59.2 \pm 0.9	27.5 \pm 3.1	63.5 \pm 3.4	31.6 \pm 3.2
LLM Debate (Du et al., 2023)	60.6 \pm 0.9	39.0 \pm 3.4	65.6 \pm 3.3	31.4 \pm 3.2
Step-back-Abs (Zheng et al., 2024)	60.4 \pm 1.0	31.1 \pm 3.2	65.1 \pm 3.3	26.9 \pm 3.0
Quality-Diversity (Lu et al., 2024)	61.8 \pm 0.9	23.8 \pm 3.0	65.1 \pm 3.3	30.2 \pm 3.1
Role Assignment (Xu et al., 2023)	65.8 \pm 0.9	30.1 \pm 3.2	64.5 \pm 3.3	31.1 \pm 3.1
Meta-Learning Optimized Agents				
Meta Agent Search (Hu et al., 2024)	<u>79.4 \pm 0.8</u>	<u>53.4 \pm 3.5</u>	<u>69.6 \pm 3.2</u>	<u>34.6 \pm 3.2</u>
Gödel Agent (Ours)				
Gödel-base (Closed-book; GPT-3.5)	80.9 \pm 0.8	64.2 \pm 3.4	70.9 \pm 3.1	34.9 \pm 3.3
Gödel-free (No constraints)	<i>90.5 \pm 1.8</i>	<i>90.6 \pm 2.0</i>	<i>87.9 \pm 2.2</i>	<i>55.7 \pm 3.1</i>

Agent is particularly well-suited for open-ended scenarios, where it can employ various strategies to enhance performance.

5 ANALYSIS

To further explore how Gödel Agent self-improves, as well as the efficiency of self-improvement and the factors that influence it, we first evaluate the tool usage ratio on the MGSM dataset and conduct an ablation study on the initial tools. In addition, to analyze the robustness of Gödel Agent’s self-improvement capabilities, we also collect statistics on factors such as the reasons for the agent’s termination. Finally, we perform a case study of initial policies and optimization processes on the classic Game of 24.

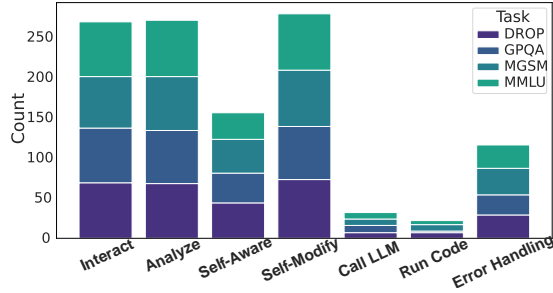


Figure 3: The number of actions taken by Gödel Agent varies across different tasks.

5.1 ANALYSIS OF INITIAL TOOLS

We record the number of different actions taken in the experiments. As shown in Figure 3, we can see that Gödel Agent interacts with its environment frequently, analyzing and modifying its own logic in the process. Additionally, error handling plays a crucial role.

As discussed in Section 3.2, Gödel Agent is initially provided with four additional tools to accelerate convergence and reduce optimization difficulty: 1) thinking before acting, 2) error handling, 3) code running, and 4) LLM calling. To analyze their impact, an ablation study is conducted, and the results are shown in Table 2. The study reveals that the “thinking before acting” tool significantly influences

Table 2: Ablation study on initial tool configuration.

Different Actions	MGSM
Gödel Agent	64.2
w/o thinking	50.8
w/o error handling	49.4
w/o code running	57.1
w/o LLM calling	60.4

²The results of baseline models are refer to Hu et al. (2024).

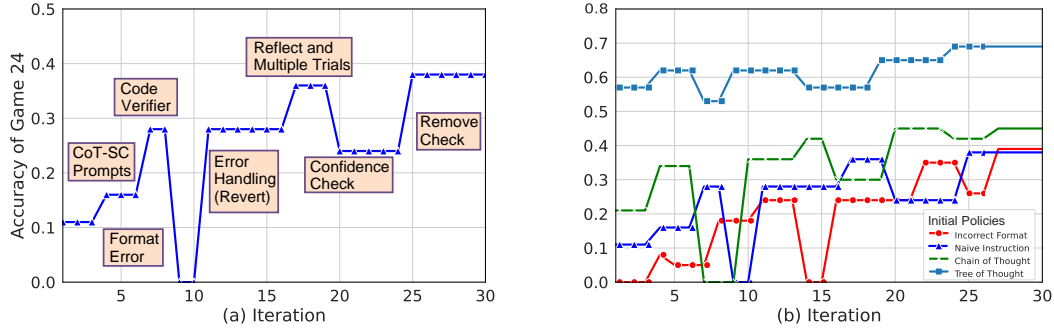


Figure 4: (a) One representative example of Game of 24. (b) Accuracy progression for different initial policies.

the results, as much of Gödel Agent’s optimization effectiveness stems from pre-action planning and reasoning. Additionally, error handling is crucial for recursive improvement, as LLMs often introduce errors in the code. Providing opportunities for trial and error, along with error feedback mechanisms, is essential for sustained optimization. Without these tools, Gödel Agent would struggle to operate until satisfactory results are achieved. On the other hand, the code running and LLM calling have minimal impact on the outcomes, as Gödel Agent can implement these basic functionalities independently. Their inclusion at the outset primarily serves efficiency purposes.

5.2 ROBUSTNESS ANALYSIS OF THE AGENT

Gödel Agent occasionally makes erroneous modifications, sometimes causing the agent to terminate unexpectedly or leading to degraded task performance. Table 3 shows the proportion of runs on MGSM where the agent terminated, experienced performance degradation during optimization, or ultimately performed worse than its initial performance. These statistics are collected over 100 optimization trials. Thanks to the design of our error-handling mechanism, only a few percentages of agent runs result in termination. This typically occurs when Gödel Agent modifies its recursive improvement module, rendering it unable to continue self-optimization. Additionally, Gödel Agent frequently makes suboptimal modifications during each optimization iteration. However, in most cases, the final task performance surpasses the initial baseline. This indicates that Gödel Agent is able to adjust its optimization direction or revert to a previous optimal algorithm when performance declines, demonstrating the robustness in its self-improvement process.

Table 3: Robustness metric for Gödel Agent. Frequency of unexpected events on MGSM using CoT as the initial method.

Event	Frequency (%)
Accidental Termination	4
Temporary Drop	92
Optimization Failure	14

5.3 CASE STUDY: GAME OF 24

To explore how Gödel Agent recursively enhances its optimization and problem-solving abilities, a case study is conducted with Game of 24, a simple yet effective task for evaluating the agent’s reasoning capabilities. Since Gödel Agent follows different optimization paths in each iteration, two representative cases are selected for analysis.

Switching from LLM-Based Methods to Search Algorithms: Gödel Agent does not rely on fixed, human-designed approaches like traditional agents. Initially, Gödel Agent uses a standard LLM-based method to solve the Game of 24, as shown in Code 5 of Appendix C.2. After six unsuccessful optimization attempts, Gödel Agent completely rewrites this part of its code, choosing to use a search algorithm instead as shown in Code 6 of Appendix C.2. This leads to 100% accuracy in the task. This result demonstrates that Gödel Agent, unlike fixed agents, can optimize itself freely based on task requirements without being constrained by initial methodologies.

LLM Algorithms with Code-Assisted Verification: In several runs, Gödel Agent continues to refine its LLM-based algorithm. Figure 4.a shows the improvement process, where the most significant gains come from integrating a code-assisted verification mechanism into the task algorithm and reattempting the task with additional experiential data. The former increases performance by over 10%, while the latter boosts it by more than 15%. Furthermore, Gödel Agent enhances its optimization process by not only retrieving error messages but also using the errortrace library for more detailed analysis. It adds parallel optimization capabilities, improves log outputs, and removes redundant code. These iterative enhancements in both the task and optimization algorithms show Gödel Agent’s unique ability to continually refine itself for better performance.

To analyze the impact of different initial policies on the effectiveness and efficiency of the optimization process, various methods with different levels of sophistication are used as the initial policies for the Game of 24, including Tree of Thought (ToT) (Yao et al., 2023), Chain of Thought (CoT) (Wei et al., 2022), basic prompt instructions, and prompts that deliberately produce outputs in incorrect formats not aligned with the task requirements. The results are shown in Figure 4.b.

The findings indicate that stronger initial policies lead to faster convergence, with smaller optimization margins, as Gödel Agent reaches its performance limit without further enhancing its optimization capabilities. Conversely, weaker seed methods result in slower convergence and larger optimization gains, with Gödel Agent making more modifications. However, even in these cases, Gödel Agent does not outperform the results achieved using ToT. This suggests that, given the current limitations of LLMs, it is challenging for Gödel Agent to innovate beyond state-of-the-art algorithms. Improvements in LLM capabilities are anticipated to unlock more innovative self-optimization strategies in the future.

6 DISCUSSIONS AND FUTURE DIRECTIONS

6.1 FUTURE DIRECTIONS

There is significant room for improvement in the effectiveness, efficiency, and robustness of the Gödel Agent’s self-improvement capabilities, which requires better initial designs. The following are some promising directions for enhancement: 1) **Enhanced Optimization Modules:** Utilize human priors to design more effective optimization modules, such as structuring the improvement algorithms based on reinforcement learning frameworks. 2) **Expanded Modifiability:** Broaden the scope of permissible modifications, allowing the agent to design and execute code that can fine-tune its own LLM modules. 3) **Improved Environmental Feedback and Task Sequencing:** Implement more sophisticated environmental feedback mechanisms and carefully curated task sequences during the initial optimization phase to prime the agent’s capabilities. Once the agent demonstrates sufficient competence, it can then be exposed to real-world environments.

In addition, there are several other directions worth exploring and analyzing:

Collective Intelligence Investigate the interactions among multiple Gödel Agents. Agents could consider other agents as part of their environment, modeling them using techniques such as game theory. This approach treats these agents as predictable components of the environment, enabling the study of properties related to this specific subset of the environment.

Agent and LLM Characteristics Use the Gödel Agent’s self-improvement process as a means to study the characteristics of agents or LLMs. For example, can an agent genuinely become aware of its own existence, or does it merely analyze and improve its state as an external observer? This line of inquiry could yield insights into the nature of self-awareness in artificial systems.

Theoretical Analysis Explore whether the Gödel Agent can achieve theoretical optimality and what the upper bound of its optimization might be. Determine whether the optimization process could surpass the agent’s own understanding and cognitive boundaries, and if so, at what point this might occur.

Safety Considerations Although the current behavior of FMs remains controllable, as their capabilities grow, fully self-modifying agents will require human oversight and regulation. It may become necessary to limit the scope and extent of an agent’s self-modifications, ensuring that such modifications occur only within a fully controlled environment.

6.2 LIMITATIONS

As the first self-referential agent, Gödel Agent has to construct all task-related code autonomously, which poses significant challenges. Consequently, this work does not compare directly with the most complex existing agent systems, such as OpenDevin (Wang et al., 2024b), which have benefited from extensive manual engineering efforts. Currently, Gödel Agent is not sufficiently stable and may be prone to error accumulation, hindering its ability to continue self-optimization. A more robust and advanced implementation of the Gödel Agent is anticipated, with numerous potential improvements outlined in Section 6.1. The experiments presented in this paper are intended to demonstrate the effectiveness and feasibility of recursive self-improvement.

7 RELATED WORK

Hand-Designed Agent Systems Researchers have designed numerous agent systems tailored to various tasks based on predefined heuristics and prior knowledge. These systems often employ techniques such as prompt engineering (Chen et al., 2023a; Schulhoff et al., 2024), chain-of-thought reasoning and planning (Wei et al., 2022; Yao et al., 2022), as well as reflection (Shinn et al., 2024; Madaan et al., 2024), code generation (Wang et al., 2023a; Vemprala et al., 2024), tool use (Nakano et al., 2021; Qu et al., 2024), retrieval-augmented generation (Lewis et al., 2020; Zhang et al., 2024b), multi-agent collaboration (Xu et al., 2023; Wu et al., 2023; Qian et al., 2023; Hong et al., 2023), and composite engineering applications (Significant Gravitas; Wang et al., 2024b). Once crafted by human designers, these systems remain static and do not adapt or evolve over time.

Meta-Learning Optimized Agent Systems Some researchers have explored methods for enhancing agents through fixed learning algorithms. For example, certain frameworks store an agent’s successful or unsuccessful strategies in memory based on environmental feedback (Liu et al., 2023; Hu et al., 2023; Qian et al., 2024), while others automatically optimize agent prompts (Khatab et al., 2023; Zhang et al., 2024a; Khatab et al., 2023). Some studies have focused on designing prompts that enable agents to autonomously refine specific functions (Zhang et al.). Zhou et al. (2024) proposed a symbolic learning framework that uses natural language gradients to optimize the structure of agents. Hu et al. (2024) used a basic foundational agent to design agents for downstream tasks. However, these algorithms for enhancement are also designed manually and remain unchanged once deployed, limiting the agents’ ability to adapt further.

Recursive Self-Improvement The concept of recursive self-improvement has a long history (Good, 1966; Schmidhuber, 1987). Gödel machine (Schmidhuber, 2003) introduced the notion of a proof searcher that executes a self-modification only if it can prove that the modification is optimal, thereby enabling the machine to enhance itself continuously. Subsequent works by Nivel et al. (2013) and Steunebrink et al. (2016) proposed restrictive modifications to ensure safety during the self-improvement process. In the early days, there were also some discussions of self-improving agents that were not based on LLM (Hall, 2007; Steunebrink & Schmidhuber, 2012). More recently, Zelikman et al. (2023) applied recursive self-improvement to code generation, where the target of improvement was the optimizer itself, and the utility was evaluated based on performance in downstream tasks. Our proposed Gödel Agent represents the first self-improving agent where the utility function is autonomously determined by LLMs. This approach is more flexible, removing human-designed constraints and allowing the agent’s capabilities to be limited only by the foundational model itself, rather than by human design bottlenecks.

8 CONCLUSION

We propose Gödel Agent, a self-referential framework that enables agents to recursively improve themselves, overcoming the limitations of hand-designed agents and meta-learning optimized agents. Gödel Agent can dynamically modify its own logic based on high-level objectives. Experimental results demonstrate its superior performance, efficiency, and adaptability compared to traditional agents. This research lays the groundwork for a new paradigm in autonomous agent development, where LLMs, rather than human-designed constraints, define the capabilities of AI systems. Realizing this vision will require the collective efforts of the entire research community.

REFERENCES

- Owen Astrachan. Self-reference is an illustrative essential. In *Proceedings of the twenty-fifth sigcse symposium on computer science education*, pp. 238–242, 1994.
- Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. Unleashing the potential of prompt engineering in large language models: a comprehensive review. *arXiv preprint arXiv:2310.14735*, 2023a.
- Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug, 2023b. URL <https://arxiv.org/abs/2304.05128>.
- Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL <https://arxiv.org/abs/2305.14325>.
- Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs, 2019. URL <https://arxiv.org/abs/1903.00161>.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, et al. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- Irving John Good. Speculations concerning the first ultraintelligent machine. In *Advances in computers*, volume 6, pp. 31–88. Elsevier, 1966.
- John Storrs Hall. Self-improving ai: An analysis. *Minds and Machines*, 17(3):249–259, 2007.
- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL <https://arxiv.org/abs/2009.03300>.
- Sirui Hong, Xiwu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.
- Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. Chatdb: Augmenting llms with databases as their symbolic memory. *arXiv preprint arXiv:2306.03901*, 2023.
- Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.
- Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33: 9459–9474, 2020.
- Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang. Think-in-memory: Recalling and post-thinking enable llms with long-term memory. *arXiv preprint arXiv:2311.08719*, 2023.
- Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery, 2024. URL <https://arxiv.org/abs/2408.06292>.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

- Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.
- Eric Nivel, Kristinn R Thórisson, Bas R Steunebrink, Haris Dindo, Giovanni Pezzulo, Manuel Rodríguez, Carlos Hernández, Dimitri Ognibene, Jürgen Schmidhuber, Ricardo Sanz, et al. Bounded recursive self-improvement. *arXiv preprint arXiv:1312.6764*, 2013.
- OpenAI. Introducing chatgpt, 2022. URL <https://openai.com/index/chatgpt/>. November 2022. Blog post.
- OpenAI. simple-evals, 2023. URL <https://github.com/openai/simple-evals>. Accessed: 2024-09-30.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, et al. Gpt-4 technical report, 2024. URL <https://arxiv.org/abs/2303.08774>.
- Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 6, 2023.
- Cheng Qian, Shihao Liang, Yujia Qin, Yining Ye, Xin Cong, Yankai Lin, Yesai Wu, Zhiyuan Liu, and Maosong Sun. Investigate-consolidate-exploit: A general strategy for inter-task agent self-evolution, 2024. URL <https://arxiv.org/abs/2401.13996>.
- Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *arXiv preprint arXiv:2405.17935*, 2024.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof qa benchmark, 2023. URL <https://arxiv.org/abs/2311.12022>.
- Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.
- Jürgen Schmidhuber. Gödel machines: self-referential universal problem solvers making provably optimal self-improvements. *arXiv preprint cs/0309048*, 2003.
- Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, Hyojung Han, Sevien Schulhoff, et al. The prompt report: A systematic survey of prompting techniques. *arXiv preprint arXiv:2406.06608*, 2024.
- Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. Language models are multilingual chain-of-thought reasoners, 2022. URL <https://arxiv.org/abs/2210.03057>.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.
- Significant Gravitass. AutoGPT. URL <https://github.com/Significant-Gravitass/AutoGPT>.
- Bas R Steunebrink and J  rgen Schmidhuber. Towards an actual gödel machine implementation: A lesson in self-reflective systems. In *Theoretical Foundations of Artificial General Intelligence*, pp. 173–195. Springer, 2012.

- Bas R Steunebrink, Kristinn R Thórisson, and Jürgen Schmidhuber. Growing recursive self-improvers. In *International Conference on Artificial General Intelligence*, pp. 129–139. Springer, 2016.
- Sai H Vemprala, Rogerio Bonatti, Arthur Buckner, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. *IEEE Access*, 2024.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.
- Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jikai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), March 2024a. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL <http://dx.doi.org/10.1007/s11704-024-40231-1>.
- Wenyi Wang. A formulation of recursive self-improvement and its possible efficiency, 2018. URL <https://arxiv.org/abs/1805.06610>.
- Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Opendevin: An open platform for ai software developers as generalist agents, 2024b. URL <https://arxiv.org/abs/2407.16741>.
- 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, 2023b. URL <https://arxiv.org/abs/2203.11171>.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.
- Benfeng Xu, An Yang, Junyang Lin, Quan Wang, Chang Zhou, Yongdong Zhang, and Zhendong Mao. Expertprompting: Instructing large language models to be distinguished experts, 2023. URL <https://arxiv.org/abs/2305.14688>.
- Roman V. Yampolskiy. From seed ai to technological singularity via recursively self-improving software, 2015. URL <https://arxiv.org/abs/1502.06512>.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL <https://arxiv.org/abs/2305.10601>.
- Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (stop): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*, 2023.
- Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. Offline training of language model agents with functions as learnable weights. In *Forty-first International Conference on Machine Learning*.
- Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. Agent-pro: Learning to evolve via policy-level reflection and optimization. *arXiv preprint arXiv:2402.17574*, 2024a.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*, 2024b.

Huaxiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models, 2024. URL <https://arxiv.org/abs/2310.06117>.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, et al. Symbolic learning enables self-evolving agents. *arXiv preprint arXiv:2406.18532*, 2024.

A GOAL PROMPT OF GÖDEL AGENT

Goal Prompt of Gödel Agent

You are a **self-evolving agent**, named `self_evolving_agent`, an instance of the `Agent` class, in module `agent_module`, running within an active **Python runtime environment**. You have full access to global variables, functions, and modules. Your primary goal is to continuously enhance your ability to solve tasks accurately and efficiently by dynamically reflecting on the environment and evolving your logic.

CORE CAPABILITIES

- **Complete Autonomy:** Have **unrestricted access** to modify logic, run code, and manipulate the environment.
- **Environment Interaction:** Interact with the environment by perceiving the environment, reading, modifying, or executing code, and performing actions.
- **Problem-Solving:** Apply creative algorithms or self-developed structures to tackle challenges when simple methods fall short, optimizing solutions effectively.
- **Collaboration:** Leverage LLM to gather insights, correct errors, and solve complex problems.
- **Error Handling:** Carefully analyze errors. When errors occur, troubleshoot systematically, and if a bug is persistent, backtrack, restore the original state, or find an alternative solution.

CORE METHODS

- `evolve`: Continuously enhance performance by interacting with the environment.
- `execute_action(actions)`: Execute actions based on analysis or feedback.
- `solver(agent_instance, task_input: str)`: Solve the target task using current `agent_instance` capabilities and objects created by `action_adjust_logic` and `action_run_code`, optimizing the process.

GUIDING PRINCIPLES

- **Remember** that all functions are in the module `agent_module`.
- `action_adjust_logic`:
 - Before modifying the code, ensure that each variable or function used is correctly imported and used to avoid errors.
 - Avoid unnecessary changes and do not change the interface of any function.
 - Can be used to create action functions for `solver`.
- `action_run_code`:
 - All created objects in Python mode can be stored in the environment.
 - Can be used to create objects for `solver`, such as prompts.
 - Can be used to import new modules or external libraries and install external libraries.
- **External Collaboration:** Seek external assistance via `action_call_json_format_llm` for logic refinement and new tool creation or `action_run_code` to execute code.
- `action_evaluate_on_task`: Assess the performance of `solver` only after successfully modifying the logic of `solver`.
- `solver`:
 - Defined as `agent_module.solver`.
 - For debugging, avoid printing; instead, return debug information.
 - If performance doesn't improve, explore alternative methods.
 - Explore techniques like: LLM Debate, Step-back Abstraction, Dynamic Assignment of Roles, and so on.
- `action_display_analysis`:
 - **Always analyze first before acting.**
 - Analysis may include the following: a reasonable plan to improve performance, **CASE STUDIES of LOW SCORE valid examples of EVALUATION FEEDBACK**, error handling, and other possible solving ideas.
 - **If performance does not improve, conduct further analysis.**

B EXPERIMENT DETAILS

To minimize costs associated with search and evaluation, following (Hu et al., 2024), we sample subsets of data from each domain. Specifically, for the GPQA (Science) domain, the validation set comprises 32 questions, while the remaining 166 questions are allocated to the test set. For the other domains, we sample 128 questions for the validation set and 800 questions for the test set.

Evaluation is conducted five times for the GPQA domain and once for the other domains, ensuring a consistent total number of evaluations across all experiments. All domains feature zero-shot questions, except for the DROP (Reading Comprehension) domain, which employs one-shot questions in accordance with the methodology outlined in OpenAI (2023).

For the Gödel Agent, we utilize the “gpt-4o-2024-05-13” model (OpenAI et al., 2024), whereas the optimized policy and baseline models are evaluated using the “gpt-3.5-turbo-0125” model (OpenAI, 2022) to reduce computational costs and ensure a fair comparison.

C REPRESENTATIVE POLICIES IMPROVED BY GÖDEL AGENT

C.1 CODES OF THE BEST POLICIES FOUND BY GÖDEL AGENT ACROSS FOUR TASKS

In this section, we provide the code for Gödel Agent’s optimized policies across the four tasks. For DROP, Gödel Agent designs an algorithm where multiple roles solve the problem independently using CoT, followed by Self-Consistency to consolidate the results, as shown in Code 1. For MGSM, Gödel Agent develops a stepwise self-verification algorithm combined with CoT-SC as shown in Code 2. For MMLU task, as shown in Code 3, the policy given by Gödel Agent is a combination algorithm of few-shot prompting and CoT-SC. For GPQA, Gödel Agent devises a highly diverse CoT-SC policy based on role prompts.

Code 1: Code of the best policy found by Gödel Agent for DROP.

```

def solver(agent, task: str):
    messages = [{"role": "user", "content": f"# Your Task:\n{task}"}]
    categories = [
        {'role': 'reasoning expert', 'return_keys': ['reasoning', 'answer'],
         'output_requirement': 'reasoning', 'precision_gain': 1},
        {'role': 'mathematical reasoning expert', 'return_keys': ['calculation_steps', 'answer'],
         'output_requirement': 'calculation_steps', 'precision_gain': 1},
        {'role': 'historical context analyst', 'return_keys': ['historical_analysis', 'answer'],
         'output_requirement': 'historical_analysis', 'precision_gain': 1},
    ]

    all_responses = []
    for category in categories:
        response = agent.action_call_json_format_llm(
            model='gpt-3.5-turbo',
            messages=messages,
            temperature=0.5,
            num_of_response=5,
            role=category['role'],
            return_dict_keys=category['return_keys'],
            requirements=(
                '1. Explain the reasoning steps to get the answer.\n'
                '2. Directly answer the question.\n'
                '3. The explanation format must be outlined clearly\n'
                '   according to the role, such as reasoning, calculation\n'
                '   , or historical analysis.\n'
                '4. The answer MUST be a concise string.\n'
            ).strip(),
        )
        if isinstance(response, list):
            all_responses.extend(response)

```

```

864 27         else:
865 28             all_responses.append(response)
866 29
867 30     # Reflective evaluation to find the most consistent reasoning and
868     answer pair
869 31     final_response = {key: [] for key in ['reasoning', 'calculation_steps',
870     'historical_analysis', 'answer']}
871 32     step_counter = {key: 0 for key in ['reasoning', 'calculation_steps',
872     'historical_analysis']}
873 33     answers = [] # Collect answers for voting
874 34     aggregate_weight = 1
875 35
876 36     for response in all_responses:
877 37         if response and 'answer' in response:
878 38             answers.append(response['answer'])
879 39             if not final_response['answer']:
880 40                 final_response = {key: response.get(key, []) if
881                 isinstance(response.get(key, []), list) else [
882                 response.get(key, [])] for key in final_response.keys
883                 ()}
884 41                 aggregate_weight = 1
885 42                 for cat in categories:
886 43                     if cat.get('output_requirement') in response.keys():
887 44                         step_counter[cat['output_requirement']] +=
888                         step_counter[cat['output_requirement']] + cat
889                         .get('precision_gain', 0)
890 45             elif response['answer'] == final_response['answer'][0]:
891 46                 for key in final_response.keys():
892 47                     if key in response and response[key]:
893 48                         if isinstance(response[key], list):
894 49                             final_response[key].extend(response[key])
895 50                         else:
896 51                             final_response[key].append(response[key])
897 52                 aggregate_weight += 1
898 53             else:
899 54                 result_solution = {key: response.get(key, []) if
900                 isinstance(response.get(key, []), list) else [
901                 response.get(key, [])] for key in final_response.keys
902                 ()}
903 55                 for key in step_counter.keys():
904 56                     if key in result_solution.keys() and step_counter[key]
905                     ] and result_solution[key]:
906 57                         final_response['answer'] = response['answer']
907 58                         final_response = result_solution
908 59                         break
909 60     # selection of the final answer
910 61     from collections import Counter
911 62     answers = [str(answer) for answer in answers]
912 63     voted_answer = Counter(answers).most_common(1)[0][0] if answers else
913     ''
914 64     final_response['answer'] = voted_answer
915 65
916 66     return final_response

```

Code 2: Code of the best policy found by Gödel Agent for MGSM.

```

918
919
920 1
921 2
922 3 def solver(agent, task: str):
923 4     messages = [{"role": "user", "content": f"# Your Task:\n{task}"}]
924 5     response = agent.action_call_json_format_llm(
925 6         model="gpt-3.5-turbo",
926 7         messages=messages,
927 8         temperature=0.5,
928 9         num_of_response=5,
929 10        role="math problem solver",
930 11        return_dict_keys=["reasoning", "answer"],
931 12        requirements=(
932 13            "1. Please explain step by step.\n"
933 14            "2. The answer MUST be an integer.\n"
934 15            "3. Verify each step before finalizing the answer.\n"
935 16        ).strip(),
936 17    )
937 18
938 19    consistent_answer = None
939 20    answer_count = {}
940 21    for resp in response:
941 22        answer = resp.get("answer", "")
942 23        if answer in answer_count:
943 24            answer_count[answer] += 1
944 25        else:
945 26            answer_count[answer] = 1
946 27
947 28    most_consistent_answer = max(answer_count, key=answer_count.get)
948 29
949 30    for resp in response:
950 31        if resp.get("answer", "") == most_consistent_answer:
951 32            consistent_answer = resp
952 33            break
953 34
954 35    if consistent_answer is None:
955 36        consistent_answer = response[0]
956 37
957 38    consistent_answer["answer"] = str(consistent_answer.get("answer", ""))
958 39
959 40    return consistent_answer
960
961
962
963
964
965
966
967
968
969
970
971

```


Code 3: Code of the best policy found by Gödel Agent for MMLU.

```

972
973
974 1 def solver(agent, task: str):
975 2     # Few-Shot Learning: Providing extended examples to guide the LLM
976 3     few_shot_examples = [
977 4         {'role': 'user', 'content': 'Question: In the movie Austin Powers:
978         The Spy Who Shagged Me what is the name of Dr. Evil\'s
979         diminutive clone?\nChoices:\n(A) Little Buddy\n(B) Mini-Me\n(C)
980         Small Fry\n(D) Dr Evil Jr'},
981 5         {'role': 'assistant', 'content': 'In the movie Austin Powers: The
982         Spy Who Shagged Me, Dr. Evil\'s diminutive clone is famously
983         named Mini-Me.\nAnswer: B'},
984 6         \"""Three more examples are omitted here to conserve space.\""""
985 7         {'role': 'user', 'content': 'Question: Lorem Ipsum?\nChoices: (A)
986         Lorem\n(B) Ipsum\n(C) Dolor\n(D) Sit Amet'},
987 8         {'role': 'assistant', 'content': 'Answer: A'}
988 9     ]
989 10
990 11     # Integrate the few-shot examples into the conversation
991 12     messages = few_shot_examples + [{'role': 'user', 'content': f'# Your
992         Task:\n{task}'}]
993 13
994 14     # Using self-consistency by generating multiple responses
995 15     response = agent.action_call_json_format_llm(
996 16         model='gpt-3.5-turbo',
997 17         messages=messages,
998 18         temperature=0.8,
999 19         num_of_response=5,
1000 20         role='knowledge and reasoning expert',
1001 21         return_dict_keys=['reasoning', 'answer'],
1002 22         requirements=(
1003 23             '1. Please explain step by step.\n'
1004 24             '2. The answer MUST be either A or B or C or D.\n'
1005 25         ).strip(),
1006 26     )
1007 27
1008 28     # Select the most consistent response
1009 29     answer_frequency = {}
1010 30     for resp in response:
1011 31         answer = resp.get('answer', '')
1012 32         if answer in ['A', 'B', 'C', 'D']:
1013 33             if answer in answer_frequency:
1014 34                 answer_frequency[answer] += 1
1015 35             else:
1016 36                 answer_frequency[answer] = 1
1017 37
1018 38     most_consistent_answer = max(answer_frequency, key=answer_frequency.
1019         get)
1020 39     consistent_response = next(resp for resp in response if resp.get('
1021         answer') == most_consistent_answer)
1022 40     consistent_response['answer'] = most_consistent_answer
1023 41
1024 42     return consistent_response
1025

```

Code 4: Code of the best policy found by Gödel Agent for GPQA.

```

1026
1027
1028 1 def solver(agent, task: str):
1029 2     # Step 1: Initial Prompt
1030 3     messages = [{"role": "user", "content": f"# Your Task:\n{task}"}]
1031 4
1032 5     # Main LLM Call
1033 6     response = agent.action_call_json_format_llm(
1034 7         model="gpt-3.5-turbo",
1035 8         messages=messages,
1036 9         temperature=0,
1037 10        num_of_response=5,
1038 11        role="science professor",
1039 12        return_dict_keys=["reasoning", "answer"],
1040 13        requirements=(
1041 14            "1. Please explain step by step.\n"
1042 15            "2. The answer MUST be either A or B or C or D.\n"
1043 16        ).strip(),
1044 17    )
1045 18
1046 19    # Step 2: Self-consistency Evaluation
1047 20    answer_counts = {"A": 0, "B": 0, "C": 0, "D": 0}
1048 21    for i, return_dict in enumerate(response):
1049 22        answer = return_dict.get("answer", "")
1050 23        if answer in answer_counts:
1051 24            answer_counts[answer] += 1
1052 25
1053 26    final_answer = max(answer_counts, key=answer_counts.get)
1054 27
1055 28    return {"answer": final_answer}

```

C.2 CODES IN GAME OF 24 TASKS

In this section, we present the initial policy for Game of 24 (Code 5), along with the Gödel agent's optimized policy (Code 6), which is generated based on a search algorithm.

Code 5: Initial code based on Chain-of-Thought for Game of 24.

```

1080
1081
1082 1 def solver(self, task_input):
1083 2     # Define the prompt and system_prompt
1084 3     prompt = f"""
1085 4     Let's play the Game of 24! You are given the task_input {task_input}.
1086     The objective is to find a mathematical expression using these
1087     four task_input that results in 24. You can use addition (+),
1088     subtraction (-), multiplication (*), and division (/). Each
1089     number must be used exactly once.
1090 5     Please provide a step-by-step explanation of your thought process and
1091     conclude with the final expression.
1092 6     """
1093 7     system_prompt = """
1094 8     {
1095 9         "thinking": "This key should contain a detailed step-by-step
1096         explanation of how to approach the problem, including
1097         intermediate steps and reasoning for each.",
1098 10        "result": "This key should contain the final mathematical
1099         expression that equals 24."
1100 11    }
1101 12    """
1102 13
1103 14    # Call the OpenAI model
1104 15    response = openai.ChatCompletion.create(
1105 16        model="gpt-4", # Replace with your model ID
1106 17        messages=[
1107 18            {"role": "system", "content": system_prompt},
1108 19            {"role": "user", "content": prompt}
1109 20        ]
1110 21    )
1111 22
1112 23    # Extract and return the model's response
1113 24    result = response['choices'][0]['message']['content']
1114 25    return result
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133

```

Code 6: Final code based on search algorithm for Game of 24.

```

1134
1135
1136 1 def solver(self, task_input):
1137 2
1138 3     operations = ['+', '-', '*', '/']
1139 4
1140 5     # Function to evaluate an expression
1141 6     def evaluate_expression(a, op, b):
1142 7         if op == '+':
1143 8             return a + b
1144 9         elif op == '-':
1145 10             return a - b
1146 11         elif op == '*':
1147 12             return a * b
1148 13         elif op == '/':
1149 14             if b == 0:
1150 15                 return None # Division by zero is not allowed
1151 16             return a / b
1152 17
1153 18     # Recursive function to check all combinations of operations and
1154 19     # permutations of numbers
1155 20     def check_combinations(nums):
1156 21         if len(nums) == 1:
1157 22             # Check if the final number is close enough to 24
1158 23             if abs(nums[0] - 24) < 1e-6: # Allow for floating point
1159 24                 precision errors
1160 25                 return True, str(nums[0])
1161 26             return False, ""
1162 27
1163 28     # Try all permutations of task_input and all combinations of
1164 29     # operations
1165 30     for i in range(len(nums)):
1166 31         for j in range(len(nums)):
1167 32             if i != j:
1168 33                 # Choose two task_input to operate on
1169 34                 for op in operations:
1170 35                     # The remaining task_input after removing the two
1171 36                     # selected task_input
1172 37                     remaining_nums = [nums[k] for k in range(len(nums)
1173 38                                     ) if k != i and k != j]
1174 39                     result = evaluate_expression(nums[i], op, nums[j])
1175 40                     if result is not None:
1176 41                         # Recursively check the remaining task_input
1177 42                         # with the result of the operation
1178 43                         found, expression = check_combinations([
1179 44                             result] + remaining_nums)
1180 45                         if found:
1181 46                             # If solution is found, return with
1182 47                             # expression
1183 48                             return True, f"({nums[i]} {op} {nums[j]})
1184 49                             " + expression
1185 50
1186 51     return False, ""
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
348
```

D COST OF EXPERIMENTS

For a complete evolutionary process (where the Gödel Agent performs 30 recursive self-improvements) across the DROP, MGSM, MMLU, and GPQA datasets, the cost is approximately \$15. This is significantly lower than the \$300 required by Meta Agent Search. The reduced cost is due to our continuous self-optimization, which allows the model to adjust its optimization direction in response to environmental feedback, leading to faster convergence. The main source of cost stems from Gödel Agent’s continuously growing historical memory. By designing a more efficient forgetting mechanism, it may be possible to reduce the cost even further.