ADAPTIVE MULTI-AGENT POOL EVOLUTION: AN ONLINE SELECTION AND GENERATION FRAMEWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing single-agent systems typically rely on fixed policies or Bayesian posterior updates to handle diverse tasks during execution. However, fixed policies lack flexibility, while Bayesian methods incur high computational overhead and converge slowly under frequent task switching. To overcome these limitations, this paper proposes AMAPE (Adaptive Multi-Agent Pool Evolution)—an online selection and generation framework that dynamically maintains an evolvable pool of sub-agents. For each incoming task, the system generates sub-agents tailored to its characteristics and caches them in the pool for direct reuse by similar tasks in the future. The framework employs the Wilson confidence interval to monitor the global execution success rate online: when the confidence lower bound falls below a preset threshold, the system triggers the generation of a new agent by synthesizing the current task description, prior experience from similar successful tasks, and failure avoidance strategies distilled from historical failures. For agent selection, we design a composite metric that integrates task similarity with each agent's Upper Confidence Bound (UCB) score, effectively balancing the exploration of newly generated agents and the exploitation of proven high-performing ones. Experimental results demonstrate that AMAPE dynamically expands and updates the agent pool in response to evolving task distributions, improving task success rates without introducing additional execution overhead.

1 INTRODUCTION

Large Language Model (LLM)-based agents have demonstrated remarkable capabilities in automating a wide range of complex tasks, with applications spanning code generation, mathematical reasoning, data analysis, and beyond (Yang et al., 2024; Lee et al., 2026; Zhang et al., 2026a). However, most existing systems adopt a fixed single-agent architecture that applies a uniform strategy to all incoming tasks. While straightforward to deploy, this approach often fails to accommodate task-specific nuances and performs poorly when faced with diverse or dynamically changing task distributions.

To address this limitation, recent work has introduced Bayesian or genetic optimization strategies to search for better agent configurations or policies (Papamarkou et al., 2026; Wu et al., 2026). Although these methods can improve performance, they typically require extensive offline computation, making them difficult to deploy in online, real-time task execution scenarios. In settings where tasks arrive sequentially and demand immediate responses, their high computational overhead and slow convergence render them impractical.

To overcome these challenges, this paper proposes AMAPE (Adaptive Multi-Agent Pool Evolution)—an online selection and generation framework that dynamically maintains an evolvable pool of sub-agents. Unlike fixed-policy or offline-optimized approaches, AMAPE generates task-specific sub-agents on the fly and caches them in the pool for future reuse. The framework employs the Wilson confidence interval to monitor the global execution success rate online: when performance falls below a preset threshold, the system synthesizes the current task description, prior successful experiences, and failure-avoidance strategies distilled from past mistakes to generate a new sub-agent. For agent selection, we design a composite metric that integrates task similarity with each agent's Upper Confidence Bound (UCB) score, effectively balancing the exploration of newly generated agents against the exploitation of proven high-performing ones.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

The main contributions of this paper are threefold:

- We propose a novel online sub-agent generation and caching framework that dynamically adapts to changing task distributions without relying on expensive offline optimization;
- We introduce a Wilson-UCB hybrid mechanism that jointly governs agent generation (via confidence monitoring) and selection (via similarity-weighted UCB), achieving principled exploration-exploitation trade-offs;
- We demonstrate through extensive experiments that AMAPE significantly improves task success rates without incurring additional execution overhead.

2 Related work

LLM Agents and Single-Agent Systems. LLM-based agents have achieved substantial progress in areas such as personalization and recommendation (Xu et al., 2025; Chen et al., 2026), code assistance (Ning et al., 2026), security (Liu et al., 2026), and academic research (Li et al., 2025), primarily by augmenting LLMs with tools, memory, and planning capabilities. Despite these successes, most single-agent systems still rely on fixed policies that are unable to adapt to complex tasks. This rigidity becomes a significant limitation when facing diverse or dynamically changing task distributions.

Multi-Agent Systems. Building on the success of single-agent systems, research on multi-agent systems has shown that combining multiple agents can surpass the capabilities of a single agent (Zhang et al., 2024; Yang et al., 2024). Early systems such as CAMEL Li et al. (2023), ChatDev Qian et al. (2024), and MetaGPT Hong et al. (2024) rely heavily on manual configurations, including prompt engineering and communication design, which limits their adaptability.

Agent Self-Evolution. Recent research has explored the self-evolution capabilities of agents across multiple dimensions: at the prompt level, ACE Zhang et al. (2026b) and GEPA Agrawal et al. (2026) evolve prompts through structured reflection; at the memory level, MemEvolve Zhang et al. (2025) and MemMA Lin et al. (2026) improve the completeness and associative accuracy of long-term memory; at the skill level, SkillRL Xia et al. (2026) and SkillOpt Yang et al. (2026) enable autonomous iteration and controllable evolution of skills; at the tool level, TOOLLEVO Chen et al. (2025) achieves self-reflection and self-update of tool usage through active exploration and interaction with dynamic environments. However, these methods still share a common paradigm—searching offline for a single fixed architecture for a given dataset, which remains static after deployment and cannot adapt to new tasks online.

Bayesian and Genetic Optimization for Agents. A parallel line of work applies Bayesian optimization or genetic algorithms to optimize agent policies or hyperparameters. These approaches Papamarkou et al. (2026); Wu et al. (2026) treat agent design as a black-box optimization problem and can yield strong performance. However, they typically require hundreds or thousands of trial runs, incurring significant computational overhead that makes them unsuitable for online deployment. Moreover, the optimized policies remain fixed and do not evolve with changing task distributions.

In contrast to all the above methods, AMAPE adopts a fundamentally different philosophy: instead of offline searching for a one-size-fits-all system, we maintain an online pool of task-specific sub-agents, generating and caching them on demand. This design enables the system to adapt to evolving tasks in real time, leveraging historical successes while avoiding the high cost of offline search.

3 Proposed method

We propose an adaptive sub-agent pool framework that dynamically maintains a population of evolvable sub-agents to address a sequence of tasks. At each task, the framework decides whether to generate a new sub-agent or to select an existing one from the pool for task execution. This decision is guided by the Wilson confidence interval, which provides a statistically principled criterion for monitoring overall system performance. By alternating between exploration (generating new agents) and exploitation (selecting proven agents), the framework achieves robust and efficient task execution in dynamic environments. The overall procedure is illustrated in Figure 1.

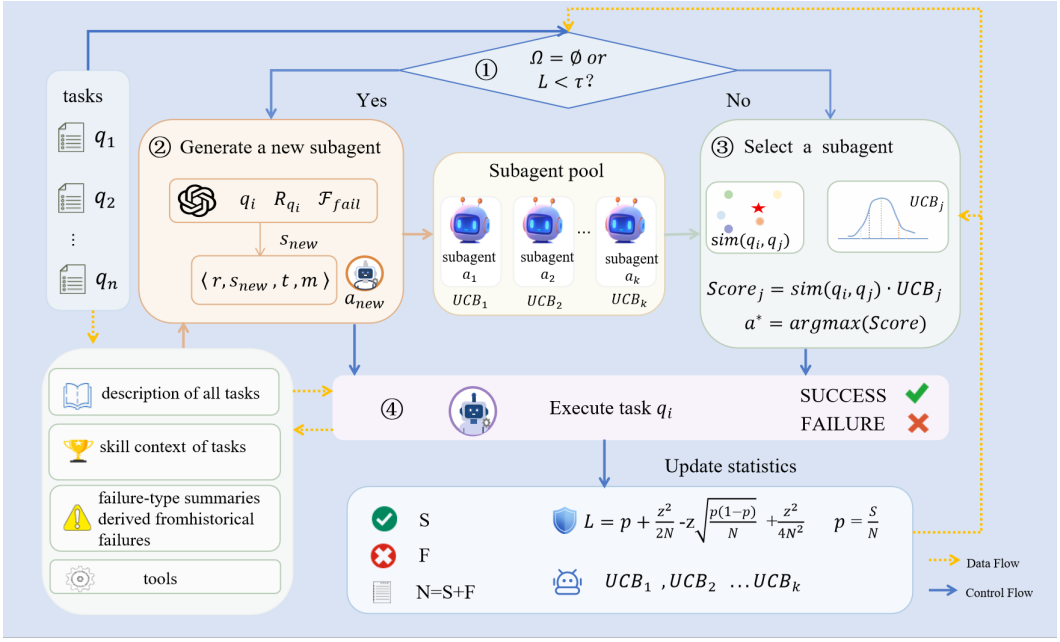


Figure 1: Overview of the adaptive sub-agent pool framework. Given a sequence of tasks, the system first decides whether to generate a new sub-agent or reuse an existing one based on the Wilson confidence lower bound, then executes the task with the selected agent, and finally updates the global statistics to guide subsequent decisions.

3.1 Problem Formulation and Initialization

Let $T = \{q_1, q_2, \ldots, q_n\}$ denote the ordered set of incoming tasks. The sub-agent pool is initially empty, i.e., $\Omega = \emptyset$. Two fixed parameters are used throughout execution: $z = 1.96$, the standard normal quantile for a 95% confidence level, and τ, a predefined threshold for the lower bound of the accuracy confidence interval. The system maintains the following cumulative counters, all initialized to zero:

- S: cumulative number of successfully completed tasks;
- F: cumulative number of failed tasks;
- $N = S + F$: total number of executed tasks;
- total_tokens = input_tokens + output_tokens: cumulative token consumption across all executed tasks.

The Wilson lower bound of accuracy, denoted by L, is computed as the 95% confidence lower bound of the observed success rate $p = S/N$:

$$L = \frac{p + \frac{z^2}{2N} - z\sqrt{\frac{p(1-p)+\frac{z^2}{4N}}{N}}}{1 + \frac{z^2}{N}}, \qquad p = \frac{S}{N}, \tag{1}$$

with $L = 0$ when $N = 0$. This bound is updated after each task execution and is compared against the threshold τ to trigger agent evolution (see Section 3.4).

Each sub-agent is defined as a quadruple:

$$a_i = \langle r,\ s_i,\ t,\ m \rangle, \tag{2}$$

where r denotes the role, s_i the skill, $t = \{t_1, \ldots, t_k\}$ the set of callable tools, and m the underlying LLM together with its configuration. Here r, t, and m are fixed across tasks, while s_i is task-specific. Accordingly, each sub-agent $a \in \Omega$ is associated with:

- $\mathcal{Q}_a^{\text{succ}}$: the set of tasks that agent a has successfully completed;
- $\mathcal{F}_{\text{fail}}$: record of failure reasons for failed tasks;
- UCB_a: the Upper Confidence Bound score of agent a, initialized to $+\infty$, which balances the empirical success rate and exploration bonus for selection.

3.2 Adaptive Generation of New Sub-agents

For each incoming task q_i, the system first determines whether the generation of a new sub-agent is necessary. This decision is triggered if either of the following conditions holds:

- The sub-agent pool is empty, i.e., $\Omega = \emptyset$;
- The Wilson confidence lower bound of the global accuracy falls below the threshold τ.

This bound provides a conservative estimate of the true accuracy, accounting for the uncertainty introduced by limited sample sizes. A low L indicates that the current pool of agents is underperforming, motivating the injection of new capabilities.

When triggered, a new sub-agent is evolved by synthesizing three sources of information: (i) the description of the current task q_i; (ii) a reference context extracted from historically successful tasks that are similar to q_i; and (iii) failure-type summaries derived from historical failures F_{fail}. Specifically, we first concatenate these three parts into a prompt:

$$p_i = BuildPrompt\big(q_i, \mathbb{R}(q_i), F_{fail}\big), \tag{3}$$

and then let the LLM π_θ generate a new skill:

$$s_{\text{new}} \sim \pi_\theta\big(\cdot \mid p_i\big), \tag{4}$$

p_i is deferred to Appendix B, and the generated skill s_{new} is presented in Appendix C. The generated skill is finally instantiated into the new sub-agent:

$$a_{\text{new}} = \langle r, s_{\text{new}},\ t,\ m \rangle. \tag{5}$$

Reference context from similar successful tasks. The term $\mathbb{R}(q_i)$ denotes the reference context extracted from historically similar successful tasks. Specifically, given the current task q_i, we first collect the set of historically successful tasks,

$$\mathcal{C} = \{\, j \mid o_j = 1 \,\}, \tag{6}$$

where $o_j \in \{0, 1\}$ indicates whether the historical task q_j was successfully completed. We then rank the candidates by their similarity to q_i and select the top-K most similar ones:

$$\mathcal{S}_K(q_i) = TopK_{j \in \mathcal{C}}\ sim(q_i, q_j), \tag{7}$$

where $sim(\cdot, \cdot) : \mathcal{Q} \times \mathcal{Q} \to [0, 1]$ is a task similarity function over the task space $\mathcal{Q}$. For each retrieved task $q_j \in \mathcal{S}_K(q_i)$, we construct a fragment by concatenating three textual components:

$$\phi(q_i, q_j) = \text{str}\big(sim(q_i, q_j)\big) \oplus q_j \oplus s_j, \tag{8}$$

where $\text{str}(\cdot)$ converts a scalar similarity score into text, q_j denotes the retrieved task itself, s_j is the skill context associated with q_j, and $\oplus$ denotes textual concatenation. The reference context is then obtained by concatenating the fragments of the retrieved tasks:

$$\mathbb{R}(q_i) = \bigoplus_{j \in \mathcal{S}_K(q_i)} \phi(q_i, q_j). \tag{9}$$

Pool update and execution. The newly generated agent is then incorporated into the pool:

$$\Omega \leftarrow \Omega \cup \{a_{\text{new}}\}, \tag{10}$$

and is subsequently used to execute q_i.

3.3 Selection from Existing Pool

If neither generation condition is met, the framework selects the most suitable agent from the existing pool Ω to execute the current task. The selection criterion balances each agent's historical competence with its relevance to the task at hand. For each agent $a \in \Omega$, we define a composite score:

$$\Phi(a) = \underbrace{\max_{q \in \mathcal{Q}_a^{\text{succ}}} sim(q_i, q)}_{\text{relevance}} \times \underbrace{\text{UCB}_a}_{\text{competence}}, \tag{11}$$

The first term captures how relevant the agent's prior successful experiences are to the current task, while the second term reflects the agent's overall reliability. The selected agent is then given by:

$$a^* = \arg\max_{a \in \Omega} \Phi(a). \tag{12}$$

This formulation naturally encourages the selection of agents that have both strong historical performance and relevant expertise, while the UCB component prevents overcommitment to agents with limited but successful histories.

3.4 Execution and Online Updates

We use a^s to denote the selected or newly generated agent, and employ a^s to execute the task q_i. Upon completion, if the execution succeeds, $S \leftarrow S + 1$, and q_i is appended to $\mathcal{Q}_{a^s}^{\text{succ}}$; otherwise, $F \leftarrow F + 1$, and the failure reason is stored in a failure-type repository for future generation guidance. Regardless of the outcome, the system increments the total task count as $N \leftarrow N + 1$, the global accuracy p and the Wilson lower bound L are recomputed using the updated S and N, logs the token consumption of this execution for subsequent efficiency computation, and updates the UCB score of the selected agent incrementally. This ensures that the generation criterion remains responsive to the latest performance. The UCB score of the selected agent is updated incrementally:

$$\text{UCB}_{a^s} = \frac{S_{a^s}}{n_{a^s}} + \sqrt{\frac{2 \ln N}{n_{a^s}}}, \tag{13}$$

where $S_{a^s} = |\mathcal{Q}_{a^s}^{\text{succ}}|$ denotes the number of tasks successfully completed by agent a^s, and n_{a^s} is the total number of tasks executed by a^s.

3.5 Iteration and Outputs

The above procedure—comprising generation decision, agent selection or generation, task execution, and statistical updates—is repeated sequentially for all tasks $q_1, q_2, \ldots, q_n$. The algorithm terminates upon processing the final task. The framework produces the following outputs:

- per-task execution outcomes with failure annotations where applicable;
- final global accuracy p and its Wilson lower bound L;
- per-agent UCB scores and execution histories;
- cumulative token consumption.

Through the integration of Wilson-confidence-guided generation and similarity-weighted UCB selection, the proposed framework dynamically adapts its agent population to the evolving task distribution, effectively balancing the exploration–exploitation trade-off in a principled and data-driven manner. The entire workflow is detailed in Algorithm 1.

4 Experiments

4.1 Experiment Setup

Tasks and Benchmarks. We evaluate the proposed method on two representative benchmarks covering two distinct domains:

Algorithm 1 AMAPE Framework

Require: Task set $T = \{q_1, q_2, \ldots, q_n\}$, threshold τ, confidence coefficient $z = 1.96$
Ensure: Execution results, success rate p, Wilson lower bound L, UCB scores of sub-agents, total token consumption

$\Omega \leftarrow \varnothing, S \leftarrow 0,\ F \leftarrow 0,\ N \leftarrow 0, p \leftarrow 0,\ L \leftarrow 0, \text{total_tokens} \leftarrow 0$ ▷ Initialization
for each task $q_i \in T$ **do**
  if $\Omega = \varnothing$ **or** $L < \tau$ **then**
    $a^s \leftarrow a_{\text{new}}$ ▷ Eq. 5
    $\text{UCB}[a^s] \leftarrow 0, \mathcal{Q}_{a^s}^{\text{succ}} \leftarrow \varnothing, \mathcal{Q}_{a_s}^{\text{total}} \leftarrow \varnothing$
  else
    for each agent $a \in \Omega$ **do**
      Compute $\Phi(a)$ ▷ Eq. 11
    end for
    $a^s \leftarrow \arg\max_{a \in \Omega} \Phi(a)$ ▷ Eq. 12
  end if
  result $\leftarrow$ EXECUTE(a^s, q_i), total_tokens $\leftarrow$ total_tokens + COUNTTOKENS(a^s, q_i)
  if result = SUCCESS **then**
    $S \leftarrow S + 1, \mathcal{Q}_{a^s}^{\text{succ}} \leftarrow \mathcal{Q}_{a^s}^{\text{succ}} \cup \{q_i\}$
  else
    $F \leftarrow F + 1,$ FAILURETYPE$(q_i,$ failure_reason$)$
  end if
  $N \leftarrow N + 1, \mathcal{Q}_{a^s}^{\text{total}} \leftarrow \mathcal{Q}_{a^s}^{\text{total}} \cup \{q_i\}, p \leftarrow S/N$
  $L \leftarrow \dfrac{p + \frac{z^2}{2N} - z\sqrt{\frac{p(1-p)+\frac{z^2}{4N}}{N}}}{1 + \frac{z^2}{N}}, \text{UCB}[a] \leftarrow \frac{|\mathcal{Q}_a^{\text{succ}}|}{n_a} + \sqrt{\frac{2 \ln N}{n_a}}$ ▷ Eq. 1, Eq. 13
end for
return Ω, p, L, $\{\text{UCB}[a] \mid a \in \Omega\}$, total_tokens

Tool Calling and Long-Horizon Planning: RealFin-Bench. RealFin-Bench consists of a set of realistic financial workflows designed to assess the model's domain-specific understanding and task execution capability in professional settings. We adopt the evaluation protocol from GenericAgent Liang et al. (2026) to ensure fair and consistent comparisons.

Mathematical Reasoning: MATH-Bench. We select 100 Level-5 problems from the MATH dataset Hendrycks et al. (2021) , covering four typical categories: Counting & Probability, Number Theory, Prealgebra, and Precalculus. Level-5 problems are considerably more challenging than other levels, involving theorem application and complex equation derivation; we deliberately choose them to better differentiate method performance. This difficulty, however, is at the sample level—MATH-Bench remains a simple task relative to RealFin-Bench, which requires tool calling and long-horizon planning.

Baselines. We compare our method against the following baselines: (1) Vanilla, where the LLM is prompted with the raw question and generates the answer directly without chain-of-thought or self-refinement; (2) Chain-of-Thought (CoT) Wei et al. (2022) , which elicits structured, step-by-step reasoning before the final answer is generated; (3) GenericAgent (GA) Liang et al. (2026), a general-purpose, self-evolving LLM agent system, which serves as the base harness; (4) Bayesian-Agent (BA) Wu et al. (2026), a posterior-guided skill optimization framework; and (5) GenericAgent+ (GA+) , a hybrid variant that first runs GenericAgent, then re-runs the failed tasks using skill contexts guided by the Bayesian agent's posterior, while ingesting the verified trajectories from the initial GA run.

Implementation Details. In our experiments, we adopt three open-source models as backbones: (1) GLM-5.2 (2) DeepSeek-V4-Flash and (3) Qwen3.8-27B. All experiments are conducted under identical hardware configurations to ensure fair comparisons. Hyperparameter settings are provided in Appendix A.

Evaluation Metrics. To comprehensively evaluate the performance and resource efficiency of each framework, we report the following metrics: Success Rate (SR) for task accuracy, Input Tokens (InTok) , Output Tokens (OutTok) , Total Tokens (TotalTok) , and Efficiency Score (EFF) . We focus

on SR and EFF, where EFF already captures the effect of token consumption; token usage therefore needs no separate consideration. EFF is defined as the number of successful tasks per million tokens, computed as:

$$\text{EFF} = \frac{S}{TotalTok(M)}, \tag{14}$$

This metric reflects the framework's ability to solve tasks under a fixed token budget, with higher EFF values indicating more favorable token utilization.

4.2 Overall Performance

We conduct comparative experiments on the RealFin benchmark, and the results are presented in Table 1. With GLM-5.2 as the backbone model, AMAPE achieves a SR of 85%, outperforming the baseline methods by a margin of 23% to 63%. Its EFF reaches 20.01, surpassing all compared methods. Switching the backbone to DeepSeek-V4-Flash, AMAPE yields a slightly lower SR of 65%, yet still significantly outperforms all other baselines while maintaining the highest EFF. With Qwen3.8-27B as the backbone, ours reaches the highest SR of 68% and simultaneously achieves the highest EFF. These results consistently demonstrate that AMAPE can steadily improve task success rates across different backbone models while achieving high token efficiency, validating its effectiveness and generalizability.

Table 1: Performance comparison on RealFin-Bench.

Model	Method	SR (%)	InTok	OutTok	TotalTok	EFF
GLM-5.2	Vanilla	30	1.64M	152k	1.79M	6.71
	CoT	22	1.50M	139k	1.64M	5.48
	GA	52	1.56M	154k	1.71M	12.27
	BA	60	1.65M	142k	1.79M	13.41
	GA+	62	2.36M	226k	2.58M	9.69
	AMAPE (ours)	**85**	1.55M	152k	1.70M	**20.01**
DeepSeek-V4-Flash	Vanilla	5	1.33M	101k	1.43M	1.40
	CoT	8	1.34M	96k	1.44M	2.09
	GA	5	1.33M	98k	1.43M	1.40
	BA	12	1.45M	110k	1.56M	3.21
	GA+	18	1.38M	98k	1.48M	3.38
	AMAPE (ours)	**65**	1.71M	123k	1.83M	**14.19**
Qwen3.8-27B	Vanilla	28	1.45M	93k	1.54M	7.15
	CoT	8	1.68M	94k	1.78M	1.69
	GA	45	1.49M	106k	1.59M	11.29
	BA	52	1.61M	87k	1.69M	12.41
	GA+	65	2.42M	151k	2.57M	10.12
	AMAPE (ours)	**68**	1.59M	104k	1.69M	**15.94**

We further evaluate AMAPE on the MATH-bench, with results shown in Table 2. Using GLM-5.2 as the backbone, AMAPE achieves an SR of 91%, slightly below GA+'s 92% but substantially higher than all other baselines. In terms of efficiency, AMAPE attains an EFF of 130.50, outperforming GA+ (117.95) by approximately 13%. Although its EFF is slightly lower than that of BA (139.57), BA achieves 89% SR, which is inferior to AMAPE. Overall, AMAPE strikes a better balance between SR and EFF. When switching to DeepSeek-V4-Flash, AMAPE achieves an SR of 88%, outperforming other baselines by 6% to 45%. Its EFF reaches 111.49, slightly lower than BA's 129.16, yet BA's SR is only 75%, substantially behind AMAPE. With Qwen3.8-27B as the backbone, ours achieves the highest SR of 82%, outperforming BA by 7%. Although its EFF is about 6 lower than that of BA, the substantial gain in SR makes ours overall more effective.

Taken together, the experimental results across both benchmarks demonstrate AMAPE's strong capability in balancing task success rate and token efficiency. Its advantage is particularly pronounced on the more challenging RealFin benchmark, where AMAPE achieves SR of 85% and 65% and 68% under GLM-5.2 , DeepSeek-V4-Flash and Qwen3.8-27B respectively, significantly outperforming

Table 2: Performance comparison on MATH-Bench.

Model	Method	SR (%)	InTok	OutTok	TotalTok	EFF
GLM-5.2	Vanilla	80	523k	107k	630k	126.90
	CoT	82	488k	113k	601k	136.45
	GA	88	530k	116k	646k	136.32
	BA	89	530k	108k	638k	**139.57**
	GA+	**92**	634k	146k	780k	117.95
	AMAPE (ours)	91	624k	73k	697k	130.50
DeepSeek-V4-Flash	Vanilla	45	428k	86k	514k	87.53
	CoT	43	376k	73k	449k	95.68
	GA	68	424k	109k	534k	127.39
	BA	75	474k	107k	581k	**129.16**
	GA+	82	648k	165k	814k	100.74
	AMAPE (ours)	**88**	687k	74k	760k	118.41
Qwen3.8-27B	Vanilla	19	540k	78k	618k	30.76
	CoT	39	536k	113k	649k	60.05
	GA	54	683k	111k	794k	68.03
	BA	75	760k	139k	899k	**83.39**
	GA+	62	1.05M	189k	1.24M	50.00
	AMAPE (ours)	**82**	947k	113k	1.06M	77.33

all baselines while maintaining high EFF. These findings consistently indicate that AMAPE can reliably improve task success rates across different backbone models and task difficulty levels while preserving high token utilization, confirming its effectiveness, generalizability, and robustness.

4.3 ABLATION STUDY

We further analyze the impact of the threshold τ on SR, EFF, and the number of sub-agents. Figure 2 illustrates how the variation of τ affects performance on RealFin-Bench, where the orange, blue and green curves represent Qwen3.8-27B, DeepSeek-V4-Flash and GLM-5.2, respectively. Subfigures 1–3 show the impact of τ on SR, EFF, and the number of sub-agents for all three models. As shown, SR reaches its peak of 68%, 85% and 65% for Qwen3.8-27B, GLM-5.2 and DeepSeek-V4-Flash when τ is set to 0.3, 0.4 and 0.5, respectively, with corresponding EFF values of 15.9, 20 and 14.2, and sub-agent counts of 2, 3 and 4. Subfigure 3 further reveals that the number of evolved sub-agents increases as τ grows. This is because τ defines the acceptable threshold for the lower bound of the accuracy confidence interval: a higher τ makes it harder for a single sub-agent to satisfy this requirement, prompting the system to evolve more sub-agents to distribute tasks and maintain overall performance.

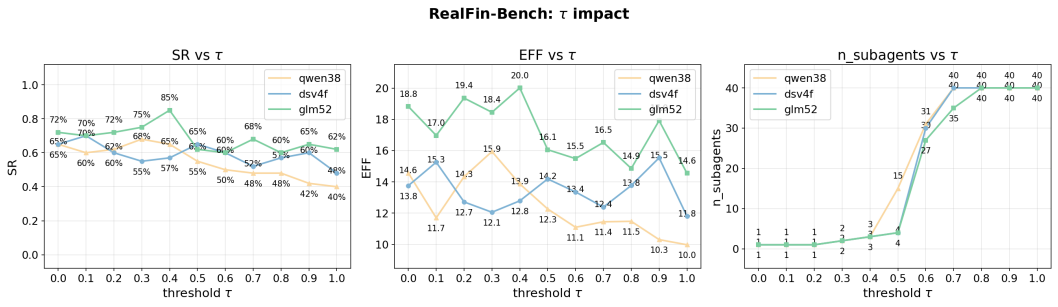


Figure 2: Effect of the threshold τ on SR, EFF, and the number of sub-agents across Qwen3.8-27B, DeepSeek-V4-Flash and GLM-5.2 on RealFin-Bench.

Unlike RealFin-Bench, SR on MATH-Bench is less sensitive to threshold variation, as shown in the first subfigure of Figure 3. As τ increases, the SR of GLM-5.2 fluctuates gently between 86% and 91%, reaching its peak of 91% at $\tau = 0.5$. At the same time, the EFF reaches 130.5 at $\tau = 0.5$, with a corresponding sub-agent count of 4. In contrast, the SR of DeepSeek-V4-Flash fluctuates between 82% and 90%; when $\tau = 0.6$, both its SR and EFF reach relatively favorable values of 88% and 118.4, respectively, with a corresponding sub-agent count of 6. Unlike DeepSeek-V4-Flash, Qwen3.8-27B exhibits certain fluctuations when the number of sub-agents is 1. As the τ increases, the SR first rises and then declines, peaking at 82% when $\tau = 0.6$; at this point, the EFF is 77.3 and the corresponding average number of sub-agents is 8.

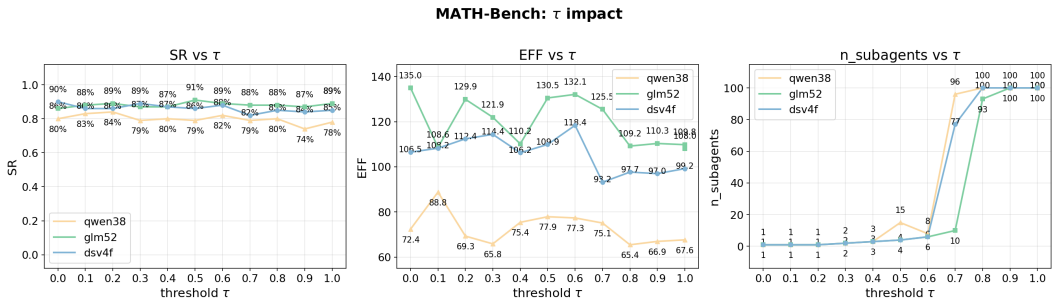


Figure 3: Effect of the threshold τ on SR, EFF, and the number of sub-agents across Qwen3.8-27B, DeepSeek-V4-Flash and GLM-5.2 on MATH-Bench.

The results in the two tables above support three conclusions. First, complex tasks (RealFin-Bench) are more sensitive to the threshold τ than simple tasks (MATH-Bench), which are largely insensitive to τ, indicating that the threshold should be tuned according to task difficulty. the number of sub-agents increases with τ, whereas the accuracy first rises and then falls, with the EFF fluctuating accordingly; this trend is pronounced on complex tasks but not on simple ones. Third, our algorithm significantly improves both SR and EFF on complex tasks without sacrificing performance on simple tasks, demonstrating robust adaptability across task difficulties.

5 Conclusion

In this paper, we address the limited capability of a single agent when handling complex problems, as well as the substantial computational overhead of Bayesian and related algorithms, and propose a multi-agent pool evolution algorithm, termed AMAPE. The algorithm achieves online optimization: it dynamically updates the Wilson lower bound of task accuracy during task execution, and based on its relationship with a preset threshold, decides whether to generate a new sub-agent or reuse an existing one. Experiments on the MATH-Bench and the agent benchmark RealFin-Bench demonstrate that AMAPE improves both the task success rate and the algorithmic efficiency. We believe that the significance of AMAPE lies not only in performance improvement, but also in offering a new online-evolution paradigm for the design of agentic systems, taking a key step toward building automated and self-evolving multi-agent systems.

6 Limitations and Future Work

The proposed AMAPE still has certain limitations. Specifically, the preset threshold in AMAPE is closely tied to the specific task: it determines when to generate a new sub-agent and when to reuse an existing one, thereby directly affecting both the task success rate and the algorithmic efficiency. However, this threshold is currently set empirically, lacking systematic theoretical guidance. For different tasks, the optimal threshold may vary depending on factors such as the task difficulty distribution and the capability distribution of sub-agents. Therefore, how to automatically search for the optimal threshold for a given task, and how to theoretically derive its optimal value, remains an open problem. We will investigate this issue in future work.

AI USE STATEMENT

Throughout the paper, the use of LLMs is solely restricted to the polishing of textual elements, such as lexical or phrasal substitutions, and does not extend beyond this scope.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2026. URL https://arxiv.org/abs/2507.19457.

Guoxin Chen, Zhong Zhang, Xin Cong, Fangda Guo, Yesai Wu, Yankai Lin, Wenzheng Feng, and Yasheng Wang. Learning evolving tools for large language models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 87140–87169, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/d8d1d325f822e07d16866780cc23deb2-Paper-Conference.pdf.

Weixin Chen, Yuhan Zhao, Jingyuan Huang, Zihe Ye, Mingxuan Ju, Tong Zhao, Neil Shah, Li Chen, and Yongfeng Zhang. MemRec: Collaborative memory-augmented agentic recommender system. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44515–44544, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.2061. URL https://aclanthology.org/2026.acl-long.2061/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Xiaodong Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *ArXiv*, abs/2103.03874, 2021. URL https://api.semanticscholar.org/CorpusID:232134851.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2024. URL https://arxiv.org/abs/2308.00352.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses, 2026. URL https://arxiv.org/abs/2603.28052.

Gen Li, Li Chen, Cheng Tang, Valdemar Švábenský, Daisuke Deguchi, Takayoshi Yamashita, and Atsushi Shimada. *Single-agent vs. Multi-agent LLM Strategies for Automated Student Reflection Assessment*, pp. 300–311. Springer Nature Singapore, 2025. ISBN 9789819681860. doi: 10.1007/978-981-96-8186-0_24. URL http://dx.doi.org/10.1007/978-981-96-8186-0_24.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 51991–52008. Curran Associates, Inc., 2023. doi: 10.52202/075280-2264. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a3621ee907def47c1b952ade25c67698-Paper-Conference.pdf.

Jiaqing Liang, Jinyi Han, Weijia Li, Xinyi Wang, Zhou Zhang, Zishang Jiang, Ying Liao, Ting-Wei Li, Ying Huang, Hao Shen, Hanyu Wu, Fangyu Guo, Keyi Wang, Zhonghua Hong, Zhiyu Lu, Lipeng Ma, Sihang Jiang, and Yanghua Xiao. Genericagent: A token-efficient self-evolving llm agent via contextual information density maximization (v1.0). *ArXiv*, abs/2604.17091, 2026. URL https://api.semanticscholar.org/CorpusID:287634286.

Minhua Lin, Zhiwei Zhang, Hanqing Lu, Hui Liu, Xianfeng Tang, Qi He, Xiang Zhang, and Suhang Wang. Memma: Coordinating the memory cycle through multi-agent reasoning and in-situ self-evolution, 2026. URL https://arxiv.org/abs/2603.18718.

Chengzhi Liu, Yichen Guo, Yepeng Liu, Yuzhe Yang, Qianqi Yan, Xuandong Zhao, Wenyue Hua, Sheng Liu, Sharon Li, Yuheng Bu, and Xin Eric Wang. Auditing agent harness safety, 2026. URL https://arxiv.org/abs/2605.14271.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, Lingjie Chen, Yanjun Zhao, Ke Yang, Bingxuan Li, Cheng Qian, Gaotang Li, Xiao Lin, Zhichen Zeng, Ruizhong Qiu, Sirui Chen, Yifan Sun, Xiyuan Yang, Ruida Wang, Rui Pan, Chenyuan Yang, Dylan Zhang, Liri Fang, Zikun Cui, Yang Cao, Pan Chen, Dorothy Sun, Ren Chen, Mahesh Srinivasan, Nipun Mathur, Yinglong Xia, Hong Li, Hong Yan, Pan Lu, Lingming Zhang, Tong Zhang, Hanghang Tong, and Jingrui He. Code as agent harness, 2026. URL https://arxiv.org/abs/2605.18747.

Theodore Papamarkou, Pierre Alquier, Matthias Bauer, Wray Buntine, Andrew Davison, Gintare Karolina Dziugaite, Maurizio Filippone, Andrew Y. K. Foong, Vincent Fortuin, Dimitris Fouskakis, Jes Frellsen, Eyke Hüllermeier, Theofanis Karaletsos, Mohammad Emtiyaz Khan, Nikita Kotelevskii, Salem Lahlou, Yingzhen Li, Fang Liu, Clare Lyle, Thomas Möllenhoff, Konstantina Palla, Maxim Panov, Yusuf Sale, Kajetan Schweighofer, Artem Shelmanov, Siddharth Swaroop, Martin Trapp, Willem Waegeman, Andrew Gordon Wilson, and Alexey Zaytsev. Position: agentic ai orchestration should be bayes-consistent, 2026. URL https://arxiv.org/abs/2605.00742.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Xiaojun Wu, Cehao Yang, Honghao Liu, Xueyuan Lin, Wenjie Zhang, Zhichao Shi, Xuhui Jiang, Chengjin Xu, Jia Li, and Jian Guo. Bayesian-agent: Posterior-guided skill evolution across llm agent harnesses, 2026. URL https://arxiv.org/abs/2606.08348.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL https://arxiv.org/abs/2602.08234.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 17577–17604. Curran Associates, Inc., 2025. doi: 10.52202/085713-0593. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/19909c36f51abc4856b4560aff3d36d6-Paper-Conference.pdf.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills, 2026. URL https://arxiv.org/abs/2605.23904.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems, 2025. URL https://arxiv.org/abs/2512.18746.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves, 2026a. URL https://arxiv.org/abs/2606.09498.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/8a94ff6f922d995d7d3f4ebf4143e442-Paper-Conference.pdf.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration, 2024. URL https://arxiv.org/abs/2408.15978.

A Experimental Setup

Across all AMAPE experiments, the Top-K retrieval parameter in Equation 7 is fixed at $K = 2$; the same LLM is used for both sub-agent generation and task execution; and all baselines are equipped with the same tool set as AMAPE.

B Prompt template for skill GENERATION

Taking RealFin as an example, we present the prompt template employed for skill generation.

You are a financial analyst. Please synthesize a concise and actionable skill context for the LLM agent that will execute the Realfin task. This guide must clearly specify the indicators to compute and the key steps to complete for the current or similar tasks, and will be prepended to the agent's prompt to steer it toward success.

Current task (to be completed):
{task_block}

Similar successful exemplars (each shows a similar task's prompt and the corresponding skill context):
{good_block}

Recent failure exemplars (mistakes to avoid):
{bad_block}

Output requirements (must be strictly followed):

Direct output: Return only the skill context itself. Do not include any thinking process, analysis, explanation, or additional remarks.

Language: The returned text must be in English.

Format: Return only a Markdown bullet list.

Conciseness: Keep it within 300 tokens — the more concise, the better.

Core action (required): Include a reminder for the model to write the final computation results to the task-specified output file.

Executability (required): Each step must name the concrete cached file paths or data sources to use (e.g. `api_cache/baostock/daily_qfq_*/sz.300XXX.csv`), the exact indicator formulas, and the output file name. The agent should be able to act on every bullet without any extra exploration.

Single-script principle (required): Structure the steps so the agent can load data, compute all indicators, and write the output in ONE code run script. Explicitly state that the agent should not explore directories or inspect manifests turn after turn — the paths are already given in this guide.

Originality: Do not copy the failure reasons or the current task prompt.

C THE EXAMPLES OF S_{NEW}

s_{new} generated in RealFin-Bench

Data Source : Load daily OHLCV for ChiNext stocks 300XXX from api_cache/baostock/; filter 2026-03-04 to 2026-03-31.

MACD : Compute EMA12/EMA26, DIFF, DEA; detect golden cross (`DIFF_{t} > DEA_{t}` and `DIFF_{t-1} <= DEA_{t-1}`).

RSI : Compute RSI(14); detect transition from < 30 to > 50.

Volume : Compute MA5 of volume; require Volume $> 2 \times$ MA5_vol on ≥ 2 distinct days.

MA Crossover : Compute MA5/MA8; detect golden cross.

Execution Run all calculations in a single Python script.

Output : Write up to 10 qualifying codes to `result.txt`, one per line.

s_{new} generated in Math-bench

-Read `task.json`; treat it as pure math — no data files, caches, or external tools.

-Identify the counting/probability structure: order matters → permutation; order irrelevant → combination.

-If direct counting is complex, try complement counting: total outcomes minus undesired outcomes.

-For probability, define sample space explicitly and ensure all outcomes are equally likely.

-Break multi-stage counts into cases; sum disjoint cases, multiply independent stages.

- Use binomial coefficients or factorial notation for clarity; simplify fractions using $\frac{a}{b}$.

-Test with small cases (e.g., n=2 or 3) to spot patterns or validate formula.

- Verify by an alternative method: recompute via different counting approach or substitution.

- Only after verification passes, place the bare final value inside `\boxed{}` — no units, no text, no $=$.

- Ensure `\boxed{}` appears in the final response text; then call finish.

- Never put intermediate or unverified values in `\boxed{}`.

- If verification fails, re-derive from scratch — do not tweak the final number.