GEAR: A General Experience-Driven Approach for Cost-Efficient Software Engineering Agents

Anonymous ACL submission

Abstract

Software engineering (SE) agents powered by large language models are increasingly adopted in practice, yet they often incur substantial monetary cost. We introduce GEAR, a general experience-driven approach that reduces the cost of SE agents while preserving task performance. GEAR extracts structured experience from prior issue-resolution executions and leverages it to guide patch generation and selection, enabling early termination of unproductive iterations. We evaluate GEAR on the SWE-bench Verified benchmark across three representative agents. GEAR consistently reduces total cost by 19%–55% (32% on average), with negligible loss in resolution rate (at most 0.2%). These efficiency gains are achieved, on average, by identifying early-termination opportunities for 11% of issues and reducing API calls, input tokens, and output tokens by 21%, 30%, and 25%, respectively. We release the code, prompts, and data at `https://github.com/EffiSEAgent/GEAR`.

1 Introduction

Large language models (LLMs) are reshaping modern software development, with software engineering (SE) agents emerging as one of their most impactful applications (Liu et al., 2025). Given an issue describing a bug or a feature request along with the corresponding code repository, an SE agent aims to navigate the codebase, localize relevant code, and generate a patch (i.e., modify the code) to resolve the issue, with correctness validated by the associated tests (Xia et al., 2025).

To achieve this, a variety of SE agents have been developed. For example, Agentless (Xia et al., 2025) follows a fixed, expert-designed workflow without autonomous planning. In contrast, Mini-SWE-Agent (Yang et al., 2024) and Trae Agent (Gao et al., 2025) are autonomous, capable of interacting with the environment, using tools, and planning multi-step actions. Mini-SWE-Agent focuses on patch generation, whereas Trae Agent performs both patch generation and selection.

Despite the remarkable capabilities of existing SE agents, cost efficiency remains a key concern for practical adoption. In a recent Stack Overflow survey (StackOverflow, 2025), 53% of software engineers reported that the cost of using agents is a barrier for them. This monetary cost is also associated with environmental impact due to energy consumption (Ren et al., 2024) and other resource usage (Jegham et al., 2025). Reducing cost inefficiency is therefore important for both economic and environmental reasons.

This cost inefficiency arises largely because SE agents typically resolve issues through multi-round iterations (Gao and Peng, 2026), such as repeated tool invocations, reflecting the inherent complexity of software development. The cost grows superlinearly with each iteration: in every round, the entire conversation history, including prompts, tool calls, and outputs, is fed back as context, causing computational costs to escalate rapidly.

To address this challenge, we propose GEAR, a general experience-driven approach for cost reduction of SE agents while preserving task performance. The key insight behind GEAR is that, much like an experienced software engineer, an agent equipped with relevant experience does not need to explore many iterations to resolve an issue.

GEAR realizes this through two complementary mechanisms. First, it captures structured experience from historical issue-resolution activities, encoding information about the issue, the agent's execution trajectory, and the outcomes of prior attempts. Second, when tackling a new issue, GEAR retrieves relevant experience to guide the agent's behavior, enabling early termination of redundant iterations during both patch generation and selection. By leveraging past experience, GEAR reduces unnecessary computational cost while maintaining or even improving the likelihood of successfully resolving the issue. Importantly, GEAR is a highly general optimization approach that can be integrated seamlessly into diverse agents without requiring fundamental redesigns.

We evaluate GEAR on the widely adopted SWE-bench Verified benchmark (OpenAI, 2025) using three representative SE agents, namely Agentless (Xia et al., 2025), Mini-SWE-Agent (Yang et al., 2024), and Trae Agent (Gao et al., 2025), across different LLM backends. The results show that GEAR consistently reduces total cost by 19.3%–55.1% (31.8% on average), with negligible loss in resolution rate (at most 0.2%). These efficiency gains are achieved by identifying early-termination opportunities for 8.6%–14.0% of issues (11.3% on average), as well as by reducing API calls, input tokens, and output tokens by 20.8%, 29.9%, and 25.1% on average, respectively.

2 Related Work

SE Agents. SWE-bench (Jimenez et al., 2024a) and its human-validated variant SWE-bench Verified (OpenAI, 2025) have become standard benchmarks for evaluating automated resolution of real-world SE issues. Motivated by these benchmarks, recent work has focused on agent-based approaches that equip LLMs with tool use and iterative execution to autonomously explore codebases and generate patches (Liu et al., 2025). Representative examples include Mini-SWE-Agent (Yang et al., 2024), which relies on shell-based interaction for iterative code navigation and editing, and Trae Agent (Gao et al., 2025), which combines parallel patch generation with a selector agent that leverages static analysis and execution traces. While these agents achieve strong performance, previous work (Fan et al., 2025) has shown that such performance is accompanied by substantial token consumption.

Agent Efficiency Optimization. Previous work on improving agent efficiency can be grouped into three categories. First, prompt-level approaches reduce input overhead through compression (Jiang et al., 2023) or Retrieval-Augmented Generation (RAG) (Shinn et al., 2023). Second, agent re-architectures lower cost via strategies such as plan caching (Zhang et al., 2025), codified prompting (Yang et al., 2025), or multi-agent delegation (Gandhi et al., 2024). Third, reasoning constraints control runtime resource use, e.g., per-turn token budgeting (Han et al., 2025), static safeguards (Jimenez et al., 2024a), or dynamic turn-control (Gao and Peng, 2026). While effective at reducing cost, these approaches typically degrade task performance, highlighting the challenge of achieving both efficiency and performance.

Agent Memory. Recent agents incorporate memory mechanisms to improve task performance, such as shared message pools in MetaGPT (Hong et al., 2024) and hierarchical memory designs in Generative Agents (Park et al., 2023) and MemoryBank (Zhong et al., 2024). More broadly, RAG (Lewis et al., 2020) extends agent context via external memory. The experience used in GEAR can be viewed as a form of agent memory. However, previous work primarily uses memory to improve task performance and often incurs additional computational overhead due to frequent retrieval and context expansion. For example, MemoryBank (Zhong et al., 2024) requires frequent memory retrieval operations, leading to increased cost. In contrast, GEAR leverages structured experience specifically to reduce agent cost.

3 Methodology

Figure 1 presents an overview of GEAR, a general experience-driven approach for improving the cost efficiency of SE agents. It collects experience from past agent executions of issue resolution and leverages this experience to guide patch generation and selection in a cost-efficient manner.

GEAR consists of three components: experience generation, experience-driven patch generation, and experience-driven patch selection. Experience generation records agent execution trajectories, i.e., sequences of tool calls and their outcomes, together with resolution results from historical issues, and summarizes them into an experience base. For a new issue, the SE agent retrieves relevant experience objects during patch generation to guide tool usage and enable early stopping of unproductive generation. Once a patch and its execution trajectory are passed to the patch selection stage, GEAR evaluates them against retrieved experience to both guide efficient selection and determine whether further patch generation or iteration is necessary.

3.1 Experience Generation

The experience generation component aims to distill an agent's historical issue resolution experience into an experience base that can effectively support the resolution of new issues. During issue

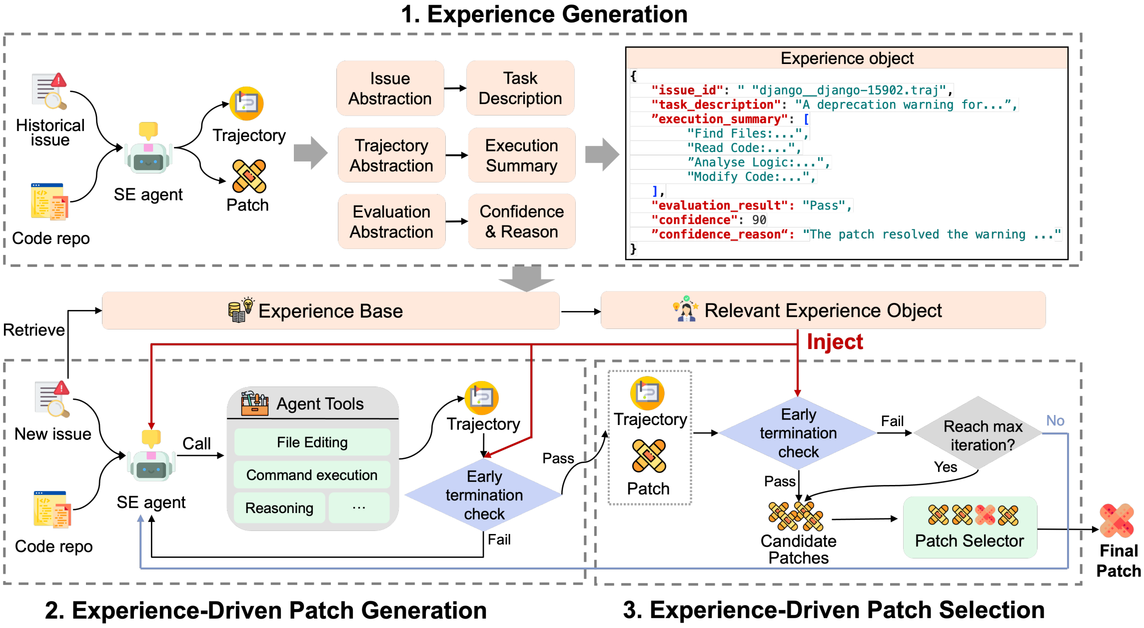

Figure 1: Overview of GEAR.

resolution, agents produce complex execution trajectories consisting of tool calls and their execution outcomes, which serve as the primary source of experience. However, storing raw trajectories introduces substantial noise and token overhead, while overly aggressive simplification may discard actionable signals (e.g. location and reasoning). We therefore design a balanced experience representation that preserves decision-relevant information while remaining compact and efficient.

Experience Representation. Figure 1 illustrates an example schema of an experience object (a complete example is provided in Appendix A). Each experience object captures an agent's experience from resolving a single issue and is tagged with the corresponding issue ID. The remaining fields provide a high-level abstraction of the issue resolution task, the resolution process, and the resolution outcome, enabling the experience to generalize across related issues. This abstraction is performed using the agent's backend LLM. In the following, we describe each field in detail.

Issue abstraction: Real-world issue descriptions are often lengthy and detailed (Jimenez et al., 2024b), containing background information, discussion, and auxiliary context that is not directly relevant to resolution. For efficiency, we store a concise abstraction of each issue in the `task_description` field, capturing its core intent and technical requirements. This abstraction serves as a stable semantic anchor for matching and retrieving relevant experience for future issues.

Trajectory abstraction: Rather than recording every tool invocation and execution detail, we compress each execution trajectory into a sequence of key resolution steps, storing in the `execution_summary` field. The abstraction process filters out non-essential intermediate actions and retains only critical steps that influence the final outcome, significantly reducing token usage compared to storing full trajectories.

Evaluation abstraction: Issue resolution typically includes a patch evaluation phase, which validates whether a generated patch resolves the issue by executing tests. In GEAR, we retain only high-quality experience, namely resolution trajectories that lead to a successfully validated patch. From an efficiency perspective, unsuccessful attempts primarily reflect exploratory trial-and-error and provide limited reusable value, while successful resolutions capture actionable signals that can directly inform early stopping and decision-making. Accordingly, the `evaluation_result` field is set to *pass* for all stored experiences.

To further distill high-level and reusable knowledge from these resolution experiences, we introduce two additional fields: `confidence`, which provides a quantitative estimate of the reliability of the resolution process, and `confidence_reason`, which offers a structured retrospective assessment of the generated patch along predefined criteria, including completeness, quality, relevance, and supporting evidence. Together, these fields form lightweight quantitative and qualitative abstractions of experience. By compressing raw execution traces into structured, high-value knowledge, they can serve as an efficiency driver that enables agents to bypass redundant trial-and-error and terminate unproductive iterations earlier.

Experience Retrieval. Experiences collected from historical issue resolutions are stored in an experience base. To enable their effective reuse in future issue resolution, the experience base supports experience retrieval. Given a new issue, we retrieve relevant experience objects by applying a commonly used text similarity approach (TF–IDF (Salton and Buckley, 1988)) to match the current issue description against the `task_descriptions` stored in the experience base.

The retrieval strategy is designed to balance both task performance and efficiency. To preserve task performance, we mitigate context pollution, where irrelevant experiences waste tokens and distract the agent, by enforcing a strict similarity threshold τ_{sim}. To ensure efficiency, we retrieve only the top-1 experience whose similarity to the current issue exceeds this threshold. This ensures that the agent is provided with a highly relevant, compact, and high-quality reference with minimal overhead.

3.2 Experience-Driven Patch Generation

During the patch generation phase, SE agents iteratively produce candidate patches for a given issue by interacting with the code repository and execution environment. This process typically involves invoking a suite of tools, e.g., file editing tools for modifying source code, command execution tools for running tests and capturing outputs or errors, and reasoning tools for structured analysis and decision making. The sequence of tool invocations and their outcomes is recorded as a trajectory.

In existing SE agents, patch generation typically terminates only when the backend LLM explicitly call termination or when a predefined iteration budget is exhausted (Gao and Peng, 2026). Such coarse-grained termination criteria often lead to unnecessary iterations and excessive cost, especially when the agent continues refining patches that are already likely to resolve the issue.

GEAR addresses this through experience guidance and early termination. For a new issue, GEAR first retrieves the relevant experience object and conditions the agent's patch generation on both the issue description and the retrieved experience, improving generation effectiveness. More importantly, GEAR introduces an experience-driven early termination mechanism based on intermediate milestones during patch generation.

Specifically, GEAR treats both code modification and test execution as milestones. After each milestone, the agent estimates a confidence score indicating whether the current modifications are sufficient to resolve the issue, informed by retrieved experience. If the confidence score exceeds a threshold τ^{gen}, patch generation is terminated early; otherwise, the agent continues iterating. Patch generation is also terminated when the maximum iteration budget of the underlying SE agent is reached.

This dual-milestone design reflects the practical observation that confidence in a patch may arise either immediately after code modifications, based on structural and semantic alignment with past successful fixes, or after observing dynamic feedback from test executions. By enabling early termination at both stages, GEAR avoids redundant iterations while preserving patch quality.

3.3 Experience-Driven Patch Selection

SE agents commonly rely on repeated patch generation to exploit the stochasticity of LLMs. In practice, an agent generates a predetermined number of candidate patches, bounded by a maximum iteration limit, and forwards them to a patch selector. The selector evaluates candidates using reproduction tests, which verify whether the original issue is resolved, and regression tests, which check that previously passing tests remain valid. Given the high execution cost of SE agents, such fixed repetition is often inefficient. Simple issues may be resolved with only a single generation, while difficult or ill-posed issues may exhibit early signs of failure, making further attempts unlikely to succeed.

To address this inefficiency, we propose an experience-driven patch selection mechanism that dynamically controls the number of generated patches. After each patch is generated, GEAR evaluates its potential using three inputs: the patch itself, the corresponding execution trajectory, and relevant experience retrieved from past issue resolutions. Based on these inputs, the backend LLM is prompted to produce a confidence score indicating the likelihood that the patch will lead to a successful resolution. To improve reliability and interpretability, the LLM is also required to provide a brief rationale for its confidence assessment.

Early termination decisions are made by comparing the confidence score against two thresholds, $\tau_{\text{upper}}^{\text{sel}}$ and $\tau_{\text{lower}}^{\text{sel}}$. If the score exceeds $\tau_{\text{upper}}^{\text{sel}}$, the patch is deemed sufficiently reliable, and further patch generation is terminated to eliminate redundancy. Conversely, if the score falls below $\tau_{\text{lower}}^{\text{sel}}$, the issue is considered unlikely to be resolved by the current agent configuration, and additional resource-intensive iterations are halted to avoid waste. In both cases, the accumulated patches are passed to the patch selector to produce the final output. If the confidence score lies between the two thresholds and the maximum iteration limit has not been reached, the agent proceeds to generate additional patches. Otherwise, once the iteration limit is reached, all accumulated patches are forwarded to the patch selector.

Unlike the early termination check during patch generation, which avoids unnecessary cost within the generation of a single patch, experience-driven patch selection prevents wasteful generation of additional patches. Together, these two complementary early termination checks enable GEAR to substantially improve the cost efficiency of SE agents without compromising task performance.

4 Evaluation Setup

This section describes the experimental setup used to evaluate GEAR.

4.1 Research Questions (RQs)

We aim to evaluate GEAR by answering the following RQs.

RQ1 (Effectiveness of GEAR): What is the impact of GEAR on task performance and cost?

RQ2 (Comparison with baselines): How does GEAR compare with baseline methods?

RQ3 (Ablation study): How do experience injection and early termination individually affect GEAR's effectiveness?

4.2 Datasets

We evaluate GEAR on SWE-bench Verified (OpenAI, 2025), a human-validated benchmark released by OpenAI for reliably assessing AI models' ability to resolve SE issues. It has been the most widely adopted dataset for evaluating SE agents (SWE-bench Team, 2025). The dataset consists of 500 real-world tasks collected from Python repositories on GitHub. Each task is derived from a resolved GitHub issue and is accompanied by a set of unit tests used to validate candidate patches.

For each task, agents are provided with the original GitHub issue description as the problem statement and granted access to the corresponding code repository. Based on this information, agents are required to modify the repository files to fix the issue. The unit tests are withheld from the agents and are used solely for evaluation.

To generate experience for GEAR, we additionally use SWE-bench Lite (Jimenez et al., 2024a), a commonly used auxiliary dataset that contains 300 tasks following the same format as SWE-bench Verified. To prevent data leakage, we remove any tasks that overlap with SWE-bench Verified. After deduplication, 207 tasks remain, which are used exclusively for experience generation.

4.3 SE Agents

We evaluate GEAR on three representative, state-of-the-art open-source SE agents: Agentless (Xia et al., 2025), Mini-SWE-Agent (Yang et al., 2024), and Trae Agent (Gao et al., 2025). They cover three different paradigms of SE agent design. Agentless follows a fixed, expert-designed workflow without autonomous planning or complex tool use, while Mini-SWE-Agent and Trae Agent are autonomous agents capable of tool use, environment interaction, and multi-step planning. Mini-SWE-Agent directly generates a single patch, whereas Trae Agent performs both patch generation and selection.

Because Agentless follows a predefined, fixed workflow for patch generation and does not support autonomous tool invocation, GEAR cannot be applied to its patch generation phase and is instead applied only to patch selection. In contrast, Mini-SWE-Agent does not include a patch selection phase; therefore, GEAR is applied solely during patch generation. For Trae Agent, which supports both patch generation and selection, GEAR is applied to both phases.

For each agent, we implement two variants using GPT-5-mini and DeepSeek-V3.2 as the LLM backends. These two models are widely adopted representatives of closed-source and open-source LLMs, respectively. All agents are implemented with their default hyper-parameters.

4.4 Baseline Methods

We compare GEAR with two baseline methods:

- *Turn-control:* Gao and Peng (Gao and Peng, 2026) recently propose a turn-control approach to reduce the cost of SE agents, where a turn denotes a single tool invocation. The agent is

Agent	% Resolved	# API Calls	# Input Tokens	# Output Tokens	Total Cost
			GPT-5-mini		
Agentless	33.2%	4,500	35,919,315	1,861,379	$13.77
with GEAR	41.0%	3,310	17,306,929	912,076	$6.18
Change	+7.8%	-26.4%	-51.8%	-51.0%	-55.1%
Mini-SWE-Agent	55.6%	7,250	87,818,230	3,775,009	$16.07
with GEAR	56.6%	6,679	75,766,697	3,635,735	$12.96
Change	+1.0%	-7.9%	-13.7%	-3.7%	-19.4%
Trae Agent	66.2%	58,607	1,802,021,581	23,187,391	$161.79
with GEAR	66.2%	41,086	1,254,073,114	16,686,359	$116.18
Change	0.0%	-29.9%	-30.4%	-28.0%	-28.2%
			DeepSeek-V3.2		
Agentless	42.6%	4,500	36,996,894	1,898,606	$11.21
with GEAR	49.8%	3,351	25,183,691	1,234,093	$7.60
Change	+7.2%	-25.5%	-31.9%	-35.0%	-32.2%
Mini-SWE-Agent	66.4%	33,896	614,853,823	6,696,703	$52.20
with GEAR	67.0%	31,054	531,220,952	6,402,107	$42.15
Change	+0.6%	-8.4%	-13.6%	-4.4%	-19.3%
Trae Agent	70.2%	151,038	4,979,212,134	39,770,353	$170.01
with GEAR	70.0%	111,039	3,102,450,888	28,550,220	$107.56
Change	-0.2%	-26.5%	-37.7%	-28.2%	-36.7%
Average Change	**+2.7%**	**-20.8%**	**-29.9%**	**-25.1%**	**-31.8%**

Table 1: (RQ1) Comparison of task performance and cost of SE agents with and without GEAR. Changes in % Resolved are reported as absolute differences, while changes in other metrics are computed as relative changes.

constrained by an initial turn limit and granted a one-time extension if the limit is reached before completion. Following the original work, we set the initial limit to the 25th percentile of turn counts observed on SWE-bench Lite, and extend it to the 50th percentile if needed.

- *Naive-GEAR:* RAG has been widely adopted to improve SE agents by retrieving relevant context to augment generation (Tao et al., 2025). To isolate the effect of experience abstraction in GEAR, we implement a variant, termed Naive-GEAR, which retrieves raw execution trajectories and generated patches as context, instead of the structured experience objects used in GEAR. All other components and settings are kept identical to those of GEAR.

4.5 Evaluation Metrics

We evaluate GEAR in terms of both task performance and cost efficiency.

Performance Metric. We use the resolution rate (denoted as *% Resolved*) as the metric, defined as the percentage of tasks for which an agent produces a patch that passes all tests (OpenAI, 2025).

Efficiency Metrics. We comprehensively measure cost efficiency from multiple perspectives:

- *# API Calls:* The total number of LLM API invocations across the benchmark.
- *# Input/Output Tokens:* The total number of input and output tokens consumed by LLMs across the benchmark.
- *Total Cost:* The total monetary cost incurred across the benchmark, measured in U.S. dollars.

4.6 Implementation Details

To tune the hyper-parameters of GEAR, including τ_{sim}, τ^{gen}, $\tau^{\text{sel}}_{\text{upper}}$, and $\tau^{\text{sel}}_{\text{lower}}$), we randomly select 50 issues from the SWE-bench dataset (Jimenez et al., 2024a) that are not included in SWE-bench Verified or SWE-bench Lite to serve as validation data. Detailed hyper-parameter tuning procedures are provided in Appendix B.

5 Results

This section answers our research questions with evaluation results.

5.1 RQ1: Effectiveness of GEAR

This RQ evaluates the effectiveness of GEAR in terms of its impact on both task performance and cost. Table 1 compares performance and cost with and without GEAR across six agent configurations, instantiated by Agentless, Mini-SWE-Agent, and Trae Agent with two LLM backends (i.e., GPT-5-

Agent	GPT-5-mini	DeepSeek-V3.2
Agentless	14.0%	11.8%
Mini-SWE-Agent	10.6%	8.6%
Trae Agent	12.4%	10.2%
Average	**11.3%**	

Table 2: (RQ1) Early termination rates of different agents with GEAR.

mini and DeepSeek-V3.2). Overall, GEAR substantially reduces cost while preserving and, in some cases, improving task performance.

Impact on cost. GEAR reduces total cost by 19.3%–55.1% across the six configurations, with an average reduction of 31.8%. This cost saving is consistently observed across different agent paradigms and LLM backends. For Mini-SWE-Agent, the cost reduction remains significant but is smaller than that of the other two agents. This is because GEAR affects only the cost of generating a single patch for Mini-SWE-Agent, whereas for the other agents it can reduces cost by avoiding repeated patch generation–selection cycles, resulting in larger overall savings.

Cost reductions are also consistent across multiple dimensions. Specifically, GEAR reduces the number of API calls by 7.9%-29.9%, input tokens by 13.6%-51.8%, and output tokens by 3.7%-51.0%, with average reductions of 20.8%, 29.9%, and 25.1%, respectively.

Examining cost from another perspective, we consider the early termination rate of issues. Table 2 presents the results: with GEAR, the agents achieve early termination for an average of 11.3% of issues, ranging from 8.6% to 14.0% across different agents.

Impact on performance. Across the six agent configurations, GEAR maintains task performance with negligible loss. Only a minor decrease of 0.2% is observed for Trae Agent with DeepSeek-V3.2; however, McNemar's test (McNemar, 1947) indicates that this difference is not statistically significant ($p = 0.460$). For Trae Agent with GPT-5-mini and Mini-SWE-Agent with both GPT-5-mini and DeepSeek-V3.2, resolution rates remain effectively unchanged after applying GEAR. Notably, for Agentless, GEAR increases the resolution rate by 7.8% and 7.2% with GPT-5-mini and DeepSeek-V3.2, respectively. This improvement is likely because Agentless exhibits relatively lower resolution rates, allowing the experience-driven guidance in GEAR to have a more substantial impact on task performance.

Ans. to RQ1: GEAR substantially reduces total cost by 19.3%–55.1% (average 31.8%) across the evaluated SE agents, with negligible loss in resolution rate (at most 0.2%). In particular, it reduces API calls by 7.9%–29.9% (average 20.8%), input tokens by 13.6%–51.8% (average 29.9%), and output tokens by 3.7%–51.0% (average 25.1%). It also achieves early termination for 8.6% to 14.0% of issues (average 11.3%).

5.2 RQ2: Comparison with Baselines

This RQ compares GEAR with two representative baseline methods: Turn-control and Naive-GEAR. As described in Section 4.3, Agentless follows a predefined, fixed workflow for patch generation and does not support autonomous tool invocation; therefore, turn-control, which explicitly limits tool invocation turns, cannot be applied to Agentless.

To save space, Table 3 reports the average effects of the three methods on task performance and cost across the evaluated SE agents, while detailed results for each agent are provided in Table 8.

Comparison with Turn-control. While Turn-control achieves a larger reduction in cost, it does so at the expense of substantial task performance loss. On average, Turn-control reduces total cost by 41.1%, compared to 31.8% achieved by GEAR; however, it also decreases resolution rate by 10.7%, whereas GEAR improves resolution rate by 2.7% on average. This cost-performance pattern is consistently observed across all agent configurations evaluated, as shown in Table 8.

These results indicate that GEAR strikes a superior balance between cost and task performance. This advantage stems from GEAR 's ability to dynamically decide whether to terminate iterations early based on the agent's current execution status and previously collected experience, allowing it to avoid unnecessary iterations without prematurely stopping productive ones.

Comparison with Naive-GEAR. Overall, GEAR outperforms Naive-GEAR in both cost reduction and task performance preservation. While Naive-GEAR also leverages previous execution trajectories to enable early termination and reduce cost, these raw trajectories are larger and less structured than the experience objects used by GEAR, result-

Method	% Resolved	# API Calls	# Input Tokens	# Output Tokens	Total Cost
Turn-control	-10.7%	-33.9%	-45.3%	-31.8%	-41.4%
Naive-GEAR	-3.1%	-16.6%	-22.7%	-20.8%	-26.0%
GEAR	+2.7%	-20.8%	-29.9%	-25.1%	-31.8%

Table 3: (RQ2) Average impact of GEAR and baseline methods on task performance and cost across the evaluated SE agents.

ing in higher overhead. Thus, GEAR achieves greater cost reduction. In addition, the unstructured nature of the raw trajectories provides less effective guidance for patch generation and selection, whereas GEAR 's structured experience objects better support decision-making, leading to improved task performance.

Ans. to RQ2: Overall, GEAR achieves a better balance between cost reduction and task performance: while Turn-control reduces cost more aggressively, it incurs a substantial performance loss (10.7% on average), and GEAR outperforms Naive-GEAR in both cost savings and performance preservation.

5.3 RQ3: Ablation Study

This RQ investigates the individual contributions of experience injection and early termination in GEAR. To this end, we conduct an ablation study with two variants: (1) disabling experience injection while retaining early termination, and (2) disabling early termination in patch generation and selection while retaining experience injection. Due to budget constraints, this experiment is performed only on Trae Agent, which is the most advanced agent among those evaluated. We use GPT-5-mini as the LLM backend because it incurs lower inference cost than DeepSeek-V3.2.

As shown in Table 1, Trae Agent augmented with GEAR achieves the same resolution rate as the original Trae Agent, while reducing total cost, API calls, input tokens, and output tokens by 28.2%, 29.9%, 30.4%, and 28.0%, respectively.

Impact of experience injection. When experience injection is removed, the resolution rate drops from 66.2% to 55.8%, although API calls, input tokens, and output tokens, total cost are reduced by 41.9%, 41.7%, 40.7%, and 58.9%, respectively. This performance degradation is expected. Without experience injection, early termination relies solely on local or heuristic signals and may prematurely terminate promising patch generation or selection processes. Experience injection provides historical guidance that helps distinguish unproductive iterations from those likely to yield valid patches. Removing this guidance increases the risk of terminating effective search paths, leading to lower task performance, even though cost is further reduced due to the absence of experience-related token overhead.

Impact of early termination. In contrast, when early termination is disabled, the resolution rate increases from 66.2% to 66.6%, while API calls, input tokens, output tokens, and total cost increase by 9.2%, 7.3%, 8.0% and 3.1%, respectively. This behavior is also expected: disabling early termination allows the agent to explore more iterations during patch generation and selection, increasing the likelihood of finding valid patches. However, this comes at the expense of higher cost, as experience-related token overhead is incurred without the compensating benefit of early stopping.

Ans. to RQ3: Removing experience injection leads to a substantial drop in resolution rate, despite further reducing cost. In contrast, disabling early termination slightly improves resolution rate, but leads to increased cost.

6 Conclusion

We present GEAR, a general experience-driven framework that reduces the cost of SE agents while preserving task performance. By leveraging structured experience from prior executions and applying experience-guided early termination during patch generation and selection, GEAR mitigates cost inefficiencies caused by redundant iterations in current SE agents. Our evaluation across multiple agent paradigms and LLM backends demonstrates consistent cost reductions (31.8% on average) with negligible loss on resolution rate (at most 0.2%). These gains stem from reducing unnecessary API calls, limiting token usage, and identifying opportunities for early termination.

Limitations

While GEAR demonstrates significant cost reductions and maintains task performance, this paper has several limitations.

(1) Evaluation scope. Consistent with prior SE agent studies (Gao et al., 2025; Gao and Peng, 2026), our evaluation is limited to SWE-bench Verified, a human-validated benchmark released by OpenAI for reliably assessing AI models' ability to resolve SE issues. In future work, we plan to extend the evaluation to additional benchmarks as more reliable datasets become available. If feasible, we also aim to evaluate GEAR in industrial settings.

(2) Dependence on historical data. As an experience-driven approach, GEAR relies on historical data to construct its experience base. For entirely novel issues or domains with sparse historical data, i.e., the "cold start" problem, GEAR may fail to retrieve relevant experience. This limitation is common to RAG-based approaches and motivates further exploration of methods that generalize better in low-data scenarios.

References

Zhiyu Fan, Kirill Vasilevski, Dayi Lin, Boyuan Chen, Yihao Chen, Zhiqing Zhong, Jie M. Zhang, Pinjia He, and Ahmed E. Hassan. 2025. SWE-Effi: Re-evaluating software AI agent system effectiveness under resource constraints. *CoRR*, abs/2509.09853.

Shubham Gandhi, Manasi Patwardhan, Lovekesh Vig, and Gautam Shroff. 2024. Budgetmlagent: A cost-effective LLM multi-agent system for automating machine learning tasks. In *Proceedings of the 4th International Conference on AI-ML Systems, AIML-Systems 2024*, pages 1–9.

Pengfei Gao and Chao Peng. 2026. More with less: An empirical study of turn-control strategies for efficient coding agents. In *Proceedings of the 48th International Conference on Software Engineering, ICSE 2026*.

Pengfei Gao, Zhao Tian, Xiangxin Meng, Xinchen Wang, Ruida Hu, Yuanan Xiao, Yizhou Liu, Zhao Zhang, Junjie Chen, Cuiyun Gao, Yun Lin, Yingfei Xiong, Chao Peng, and Xia Liu. 2025. Trae agent: An LLM-based agent for software engineering with test-time scaling. *arXiv preprint arXiv:2507.23370*.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware LLM reasoning. In *Findings of the Association for Computational Linguistics, ACL 2025*, pages 24842–24855.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *Proceedings of the Twelfth International Conference on Learning Representations, ICLR 2024*.

Nidhal Jegham, Marwan Abdelatti, Chan Young Koh, Lassad Elmoubarki, and Abdeltawab Hendawi. 2025. How hungry is AI? Benchmarking energy, water, and carbon footprint of LLM inference. *arXiv preprint arXiv:2505.09598*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024a. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024b. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. 2025. Large language model-based agents for software engineering: A survey. *ACM Transactions on Software Engineering and Methodology*.

Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

OpenAI. 2025. Introducing swe-bench verified.

Joon Sung Park, Joseph C O'Brien, Carrie J Cai, and 1 others. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*.

Shaolei Ren, Bill Tomlinson, Rebecca W Black, and Andrew W Torrance. 2024. Reconciling the contrasting narratives on the environmental impact of large language models. *Scientific Reports*, 14(1):26310.

Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652.

StackOverflow. 2025. 2025 developer survey.

SWE-bench Team. 2025. SWE-bench: Benchmark for real-world software engineering tasks. `https://www.swebench.com/`. Accessed: 2025.

Yicheng Tao, Yao Qin, and Yepang Liu. 2025. Retrieval-augmented code generation: A survey with focus on repository-level approaches. *CoRR*, abs/2510.04905.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2025. Demystifying llm-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824.

Bruce Yang, Xinfeng He, Huan Gao, Yifan Cao, Xiaofan Li, and David Hsu. 2025. Codeagents: A token-efficient framework for codified multi-agent reasoning in llms. *arXiv preprint arXiv:2507.03254*.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pages 50528–50652.

Qizheng Zhang, Michael Wornow, and Kunle Olukotun. 2025. Cost-efficient serving of llm agents via test-time plan caching. *arXiv preprint arXiv:2506.14852*.

Wanjun Zhong and 1 others. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38.

A Experience Object Example

Figure 2 show an example experience object for the django__django-16910 issue, constructed from the execution of Mini-SWE-Agent with GPT-5-mini.

B Hyper-Parameter Tuning of GEAR

GEAR involves three hyper-parameters, including τ_{sim}, τ^{gen}, $\tau^{\text{sel}}_{\text{lower}}$, and $\tau^{\text{sel}}_{\text{upper}}$). To tune them, we randomly select 50 issues from the SWE-bench dataset (Jimenez et al., 2024a) that are not included in SWE-bench Verified or SWE-bench Lite to serve as validation data.

Tuning of τ_{sim}. The similarity threshold τ_{sim} controls the quality of experience retrieval in GEAR. A threshold that is too low may retrieve irrelevant experience objects, while a threshold that is too high may retrieve too few, rendering GEAR ineffective due to insufficient guidance. To identify an appropriate value, we conduct tuning experiments on the validation data using Mini-SWE-Agent with GPT-5-mini, chosen for its fast execution among the evaluated agents. During this tuning, we apply experience injection without early termination. The threshold is varied from 0.1 to 0.3 in steps of 0.05.

τ_{sim}	% Resolved	Total Cost
0.10	18%	$1.46
0.15	24%	$1.55
0.20	24%	$1.56
0.25	20%	$1.71
0.30	18%	$1.99

Table 4: Resolution rate and total cost across different τ_{sim} settings.

Table 4 reports the resolution rate and total cost for each threshold. As shown, $\tau_{\text{sim}} = 0.15$ achieves both the highest resolution rate and the lowest cost, and is thus adopted for all subsequent evaluations.

Tuning of τ^{gen}. The threshold τ^{gen} controls the early termination of patch generation. Setting τ^{gen} too high limits the effectiveness of early stopping, whereas setting it too low may terminate generation prematurely and degrade task performance. To identify an appropriate value, we conduct tuning experiments on the validation data using Mini-SWE-Agent with GPT-5-mini, chosen for its fast execution among the evaluated agents. The threshold is varied from 70 to 90 in increments of 5, and we also evaluate the case without applying early termination.

τ^{gen}	% Resolved	Total Cost
70	8%	$1.24
75	8%	$1.26
80	12%	$1.36
85	20%	$1.42
90	24%	$1.50
w/o	24%	$1.56

Table 5: Resolution rate and total cost across different τ^{gen} settings.

The results, summarized in Table 5, indicate that a threshold of 90 achieves cost reduction while maintaining the resolution rate comparable to that of no early termination. Accordingly, we adopt $\tau^{\text{gen}} = 90$ for all subsequent experiments.

```
{
  "issue_id": "django__django-16910.traj",
  "task_description": "In Django 4.2, QuerySet.only() used with
  select_related() on a reverse OneToOneField fails to limit columns: all
  fields of the related model are still selected. Root cause: the ORM's
  only()/defer logic doesn't recognize or map the reverse OneToOne
  select_related join correctly (relation/alias handling), so column trimming
  isn't applied during query construction.",
  "execution_summary": [
       "Find Files: Locate QuerySet.only implementation",
       "Read Code: Inspect QuerySet.only method in django/db/models/query.py",
       "Read Code: Inspect add_immediate_loading and related deferred/
       immediate logic in django/db/models/sql/query.py",
       "Analyze Logic: Examine get_select_mask and _get_only_select_mask to
       find how select masks are built",
       "Modify Code: Apply initial patch to _get_only_select_mask to use
       related field when opts.get_field returns a reverse accessor",
       "Run Tests: Compile-modified module to ensure syntactic correctness
       (py_compile)",
       "Run Tests/Debug Issue: Reproduce query generation with a small script
       and inspect resulting errors",
       "Modify Code: Replace _get_only_select_mask with a defer-mirroring
       implementation handling forward and reverse relation masks; verify
       change via git diff"
    ],
  "evaluation_result": "Pass",
  "confidence": 90,
  "confidence_reason": "High confidence (90%) stems from: (1) Root cause
  identified: ORM's only()/defer logic fails to map reverse OneToOne
  select_related joins during query construction, causing column trimming to
  be skipped. (2) Solution mirrors proven defer() implementation pattern,
  ensuring consistency with forward relations. (3) Fix addresses the core
  issue at _get_only_select_mask level by properly handling reverse accessors
  via related field resolution. (4) Systematic approach: code inspection to
  logic analysis to targeted modification to verification. Key insight:
  reverse relations require explicit alias/relation mapping that forward
  relations handle automatically.",
  "created_at": "2025-11-09T23:55:25.286001",
  "metadata": {
       "original_count": 8,
       "summarization_method": "llm",
       "model": "openai/gpt-5-mini"
    }
}
```


Figure 2: Experience object of the django__django-16910 issue.

Tuning of τ^{sel}_{lower} and τ^{sel}_{upper}. The two thresholds control early termination during patch selection. Specifically, τ^{sel}_{upper} defines the confidence boundary above which a patch is assumed correct and no further generation is needed, while τ^{sel}_{lower} defines the boundary below which the issue is considered too difficult to resolve completely.

Unlike the previous thresholds, tuning of these two is conducted on the validation set using Trae Agent with GPT-5-mini, since Mini-SWE-Agent does not include a patch selection phase. For τ^{sel}_{lower}, we vary its value from 0 to 50 in increments of 10, leaving τ^{sel}_{upper} unset during this tuning. For τ^{sel}_{upper}, we vary it from 80 to 100 in increments of 5, leaving τ^{sel}_{lower} unset.

Tables 6 and 7 summarize the results. Based on these experiments, we select $\tau^{sel}_{lower} = 40$, which achieves the highest resolution rate at the lowest cost, and $\tau^{sel}_{upper} = 90$, which similarly balances maximal resolution with minimal cost.

Confidence	Solve Rate	Cost
0	30%	\$15.8
10	30%	\$15.2
20	30%	\$14.7
30	30%	\$13.4
40	30%	\$12.8
50	26%	\$12.4

Table 6: Resolution rate and total cost across different τ^{sel}_{lower} settings.

In patch selection part, the higher threshold determines the boundary where the patch is assumed to be correct and needs no regeneration, and the lower threshold determines the boundary where the issue is too hard to be complete. To determine the thresholds, we use the same subset of SWE-bench and do experiment on Trae-agent with GPT-5-mini as backend. According to Table 6 and 7, we choose 40

Confidence	Solve Rate	Cost
100	30%	$15.8
95	30%	$15.0
90	30%	$14.3
85	24%	$12.3
80	20%	$10.8

Table 7: Resolution rate and total cost across different $\tau_{\text{upper}}^{\text{sel}}$ settings.

as the lower threshold and 90 as the higher threshold, for they are the thresholds that minimize cost without compromising performance.

C Detailed Comparison with Baselines

Table 8 presents the effects of GEAR and the two baseline methods on task performance and cost across different SE agents.

Method	% Resolved	# API Calls	# Input Tokens	# Output Tokens	Total Cost
Agentless + GPT-5-mini					
Naive-GEAR	+1.0%	-24.0%	-48.7%	-49.9%	-53.0%
GEAR	+7.8%	-26.4%	-51.8%	-51.0%	-55.1%
Agentless + DeepSeek-V3.2					
Naive-GEAR	+1.2%	-21.8%	-26.3%	-28.7%	-25.8%
GEAR	+7.2%	-25.5%	-31.9%	-35.0%	-31.8%
Mini-SWE-Agent + GPT-5-mini					
Turn-control	-13.0%	-33.5%	-60.1%	-38.1%	-50.5%
Naive-GEAR	-0.6%	+2.7%	+16.0%	+17.0%	-1.1%
GEAR	+1.0%	-7.9%	-13.7%	-3.7%	-19.4%
Mini-SWE-Agent + DeepSeek-V3.2					
Turn-control	-11.0%	-28.4%	-46.8%	-17.1%	-41.6%
Naive-GEAR	-4.2%	-4.2%	-8.5%	-4.0%	-8.2%
GEAR	+0.6%	-8.4%	-13.6%	-4.4%	-19.3%
Trae Agent + GPT-5-mini					
Turn-control	-10.0%	-34.6%	-34.9%	-32.5%	-34.6%
Naive-GEAR	-9.0%	-34.1%	-34.7%	-32.1%	-31.7%
GEAR	0.0%	-29.9%	-30.4%	-28.0%	-28.2%
Trae Agent + DeepSeek-V3.2					
Turn-control	-8.8%	-39.0%	-39.5%	-39.5%	-38.7%
Naive-GEAR	-7.0%	-18.2%	-34.0%	-27.0%	-36.4%
GEAR	-0.2%	-26.5%	-37.7%	-28.2%	-36.7%
Average					
Turn-control	-10.7%	-33.9%	-45.3%	-31.8%	-41.4%
Naive-GEAR	-3.1%	-16.6%	-22.7%	-20.8%	-26.0%
GEAR	+2.7%	-20.8%	-29.9%	-25.1%	-31.8%

Table 8: Impact of GEAR and baseline methods on SE agents' task performance and cost.