DOING MORE WITH LESS TOKENS: HIERARCHICAL REINFORCEMENT LEARNING FOR EFFICIENT CODING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recently, coding agents have emerged as a dominant paradigm for real-world software engineering (SWE) scenarios, which solve complex tasks through multi-turn interactions with development environments. However, frequent interactions with environments would inevitably introduce substantial token overhead, leading to high usage costs and latency. Although recent studies have explored reducing token usage by context manipulation and interaction limits at inference time, these approaches focus on improving token efficiency while overlooking the risk of discarding task-relevant information, thus struggling to balance the trade-off between resolution rate and token efficiency. In this paper, we study a more general paradigm without suffering from the limitation, *i.e.*, training token-efficient coding agents with promising resolution performance, which is a highly-practical yet less-explored problem. To this end, we reveal two core observations in SWE scenarios: i) *Efficiency Variation*: successful resolution could be achieved with fewer tokens; ii) *Entropy Correlation*: unproductive behaviors are associated with turn-level entropy. Motivated by observations, we propose a novel reinforcement learning framework, dubbed HERO. Specifically, HERO prioritizes task resolution over token efficiency during policy optimization and encourages efficient reasoning patterns at both trajectory and turn levels. Extensive experiments on SWE-bench Verified and SWE-bench Multilingual demonstrate that HERO achieves a favorable trade-off between resolution rate and token efficiency compared with state-of-the-art coding agents and reinforcement learning methods.

1 INTRODUCTION

Recently, coding agents (Yang et al., 2024; Zhang et al., 2024; Huang et al., 2023) powered by large language models (LLMs) (Chang et al., 2024; Minaee et al., 2024) have emerged as a dominant paradigm for real-world software engineering (SWE) (He et al., 2025; Jin et al., 2024) scenarios, demonstrating strong performance across various complex tasks, including but not limited to repository-level bug fixing, feature implementation, and end-to-end software development. Specifically, coding agents first interact with environments over multiple turns through different tools, *e.g.*, search, editing, and command-line tools. By incorporating tool feedback from environments, coding agents could continually accumulate task-relevant information and thus facilitate the resolution of complex SWE tasks.

Despite the promising performance of existing coding agents, the frequent interactions with environments would inevitably introduce substantial token overhead, leading to high usage costs and latency (Fan et al., 2025). To remedy this, recent studies have explored pruning (Wang et al., 2026c) or compressing (Liu et al., 2026a; Lodha et al., 2026) the input context and limiting interaction turns (Gao & Peng, 2025) at inference time. However, such inference-time approaches focus on improving token efficiency while overlooking the risk of discarding task-relevant information, thus struggling to balance the trade-off between resolution performance and token efficiency. Different from the above inference-time designs, an alternative paradigm is to internalize both effective and efficient reasoning patterns into the agent policy through training-time designs, thereby reducing token usage without sacrificing the solution capability.

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.

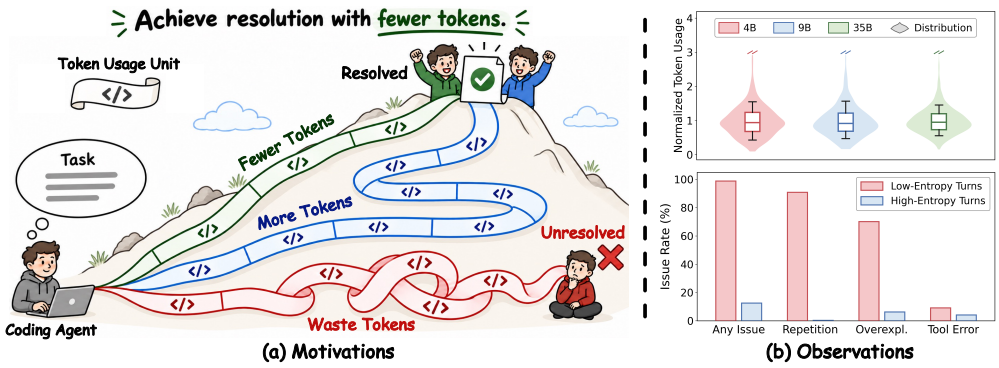


Figure 1: (a) **Motivations**: for a given SWE task, the coding agent could generate multiple reasoning trajectories with different token usages and resolution outcomes. Ideally, the goal of token-efficient coding agents is to achieve resolution with fewer tokens. (b) **Observations**: we compare token usage among successful trajectories within each task and analyze the correlation between turn-level entropy and unproductive behaviors. Note that token usage is normalized by the per-task mean over successful trajectories. As a result, one could observe that i) successful trajectories exhibit substantial variation in token usage, and ii) low-entropy turns are more prone to reasoning issues, *i.e.*, repeated tool calls, excessive exploration, and erroneous commands. Together, these observations provide guidance for learning effective and efficient reasoning patterns.

To learn effective and efficient reasoning patterns, the most promising solution might be token-efficient reinforcement learning (RL) (Arora & Zanette, 2026; Liu et al., 2026b; Xie et al., 2026), which encourages task success and penalizes token-intensive trajectories during policy optimization. However, existing methods are specifically designed for non-interactive reasoning or simple agentic tasks, which are inadequate for long-horizon SWE tasks due to the following two reasons. On the one hand, it remains under-explored whether coding agents could achieve a desirable trade-off between task success and token efficiency. On the other hand, simply penalizing token-intensive trajectories would discourage all interaction turns, including necessary reasoning steps, thereby hindering the acquisition of task-relevant information.

To specifically develop a token-efficient RL method for training coding agents, we present two core observations as illustrated in Fig. 1. To be specific, i) *Efficiency Variation*: for a given task, it is possible to achieve successful resolution through a more efficient reasoning path with fewer tokens, paving the way to develop token-efficient coding agents; ii) *Entropy Correlation*: unproductive turns could be identified based on the turn-level entropy, providing a promising avenue to learn fine-grained efficient patterns through penalizing inefficient turns.

Based on the above observations, we develop token-efficient coding agents by learning efficient reasoning patterns while preserving resolution capability. To this end, we propose a novel training framework, dubbed HiErarchical ReinfOrcement learning (HERO), which couples a hierarchical policy optimization framework with multi-granularity credit assignment. In brief, HERO consistently prioritizes task resolution over token efficiency and activates efficiency optimization once sufficient problem-solving capability is acquired. Thanks to such a hierarchical optimization objective, coding agents could achieve a favorable trade-off between resolution rate and token efficiency. Moreover, HERO adopts coarse-grained trajectory credit to encourage concise reasoning paths and fine-grained turn credit to penalize unproductive turns, thereby learning efficient reasoning patterns at multiple granularities. Experiments demonstrate that, using only 640 training tasks, HERO improves the average resolution rate from 37.4% to 42.2% across Qwen3.5 variants on SWE-bench series. Compared with GRPO, HERO yields better average improvement in resolution rate with 39.8% less token usage.

In summary, the major contributions and novelties of this work are given as follows.

- Different from existing inference-time approaches, we aim to internalize both effective and efficient reasoning patterns into the agent policy through training-time designs, thus achieving token-efficient coding agents without sacrificing resolution performance.

- We reveal two core observations regarding token-efficient coding agents in SWE scenarios, which not only demonstrate the potential to achieve a good trade-off between task success and token efficiency but also pave an avenue toward better efficiency through fine-grained optimization.
- We propose a novel method termed HERO, which couples resolution-prioritized policy optimization with multi-granularity credit assignment at trajectory and turn levels, thereby improving token efficiency while preserving resolution performance.
- Extensive experiments on SWE-bench Verified and SWE-bench Multilingual demonstrate that the proposed HERO achieves a favorable trade-off between resolution rate and token efficiency, significantly outperforming existing state-of-the-art coding agents and reinforcement learning methods.

2 Related Work

In this section, we provide a brief review of two topics highly related to this work, including coding agents and token-efficient reinforcement learning.

2.1 Coding Agents

Coding agents aim to solve real-world software engineering tasks by equipping LLMs with various tools to interact with environments. Based on how LLMs are endowed with tool-use capabilities, existing approaches could be broadly grouped into two categories: i) scaffold-based methods (Yang et al., 2024; Wang et al., 2025b; Xia et al., 2024), which design general-purpose tool commands and interaction workflows that could be applied to most widely-used LLMs; ii) training-based methods (Pan et al., 2024; Jain et al., 2025; Song et al., 2026), which enhance the ability to use the specific agent scaffold through supervised fine-tuning or reinforcement learning.

Despite their promising performance, existing coding agents focus on improving resolution rate for complex SWE tasks, while paying limited attention to the substantial token usage incurred by long-horizon interactions. Different from them, this work aims to train token-efficient coding agents that simultaneously account for reasoning efficiency and resolution performance.

2.2 Token-Efficient Reinforcement Learning

Token-efficient reinforcement learning aims to reduce the token usage of LLMs through policy optimization. According to the application scenarios, existing approaches could be broadly divided into two categories: i) reasoning-oriented methods (Arora & Zanette, 2026; Shrivastava et al., 2026; Liu et al., 2026b), which encourage concise chain-of-thought reasoning by adopting explicit length constraints or length-based rollout filtering; ii) agent-oriented methods (Wang et al., 2025a; Xie et al., 2026), which penalize excessive cumulative token usage over multiple turns and are primarily adopted in simple agentic tasks, *e.g.*, web search and multi-hop question answering.

The major differences between existing token-efficient RL methods and this work are given below. On the one hand, existing methods overlook the non-negligible unproductive turns in long-horizon SWE tasks, whereas our approach could identify and discourage unproductive behaviors and thus facilitate token efficiency in multiple interactions. On the other hand, to the best of our knowledge, our approach is one of the first works to explore simultaneously preserving resolution rate and improving token efficiency in training coding agents.

3 Method

In this section, we introduce HiErarchical ReinfOrcement learning (HERO) for training efficient coding agents. In Section 3.1, we present the goal of training token-efficient coding agents. In Section3.2, we propose hierarchical policy optimization to preserve resolution performance during efficiency optimization. In Section 3.3, we introduce multi-granularity credit assignment to learn efficient reasoning patterns at multiple granularities.

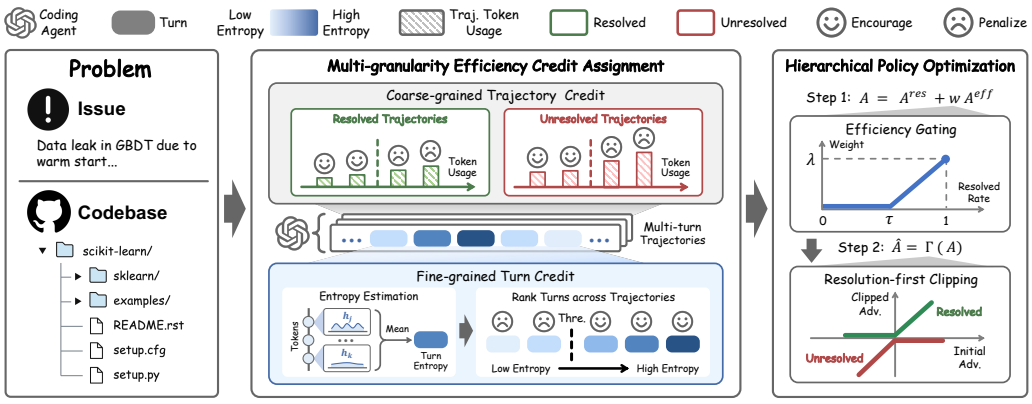


Figure 2: Overview of our method HERO. Given a swe task, the coding agent first samples multiple trajectories to form a rollout group and then computes multi-granularity advantages at trajectory and turn levels. Specifically, HERO encourages concise reasoning trajectories with the same outcome, while penalizing potentially unproductive behaviors with low turn entropy. After that, HERO adaptively activates efficiency optimization based on the resolution rate within the group and adopts resolution-first clipping to guarantee that resolved trajectories receive advantages no lower than unresolved ones.

3.1 Problem Formulation

Let $x_i = (r_i, s_i)$ denote a repository-level task in SWE scenarios, where r_i is the repository environment and s_i is the corresponding issue report. For a given r_i with s_i, the goal of coding agent $\mathcal{A}$ is to interact with environment r_i for n_i turns and then generate solution $\mathcal{A}(r_i, s_i)$ for s_i. To verify the effectiveness of the generated solution, we adopt the test-driven evaluation protocol and the corresponding resolution outcome could be derived as,

$$y_i = \mathcal{P}_i\left(\mathcal{A}(r_i, s_i)\right) \in \{0, 1\}, \tag{1}$$

where $\mathcal{P}_i(\cdot)$ indicates whether the generated solution $\mathcal{A}(r_i, s_i)$ passes all task-specific tests. As discussed in the Introduction, beyond resolution performance, we further consider the cumulative token usage during multi-turn interactions. For a given task x_i, the cumulative token usage is defined as,

$$c_i = \sum_{j=1}^{n_i} t_{i,j}, \tag{2}$$

where $t_{i,j} = t_{i,j}^{\text{in}} + t_{i,j}^{\text{out}}$ denotes the token usage at the j-th turn, with $t_{i,j}^{\text{in}}$ and $t_{i,j}^{\text{out}}$ representing the input and output tokens, respectively.

With the above formulation, we define the objective of token-efficient coding agents as maximizing the Token-efficient Resolution Score (TRS), *i.e.*,

$$\max_{\theta} \ \text{TRS}, \qquad \text{TRS} = \frac{R}{\sqrt{1 + C}}, \tag{3}$$

where $R = \frac{100}{N}\sum_{i=1}^{N} y_i$ and $C = \frac{1}{N}\sum_{i=1}^{N} c_i$ denote the average resolution rate and cumulative token usage over N tasks, respectively. Note that C is measured in millions of tokens. Clearly, a higher TRS generally indicates a more favorable trade-off between resolution rate and token usage. However, existing coding agents are typically designed to improve resolution performance, while overlooking the substantial token overhead incurred by multi-turn interactions.

As a remedy, we propose HiErarchical ReinfOrcement learning (HERO) as illustrated in Fig. 2, which couples hierarchical policy optimization over resolution and efficiency objectives with the multi-granularity efficiency credit assignment mechanism. In the following, we will elaborate on each of them.

3.2 HIERARCHICAL POLICY OPTIMIZATION

As discussed in Introduction, existing token-efficient RL methods struggle to achieve a favorable trade-off between resolution rate and token efficiency. To address this, we propose a novel hierarchical policy optimization (HPO) mechanism that improves resolution performance while reducing token usage, *i.e.*,

$$\begin{aligned} \mathcal{L}(\theta) &= -\mathbb{E}_{i\in\mathcal{G}}\left[\min\left(\rho\left(\theta\right)\widehat{A}_i, \text{clip}\left(\rho\left(\theta\right), 1-\epsilon, 1+\epsilon\right)\widehat{A}_i\right)\right], \\ \widehat{A}_i &= \Gamma(A_i), \qquad A_i = A_i^{\text{res}} + wA_i^{\text{eff}}, \end{aligned} \tag{4}$$

where $\mathcal{G}$ denotes the rollout group for the same task, $\rho(\theta)$ denotes the importance ratio between the current and rollout policies, ϵ indicates the clipping threshold. In particular, $\Gamma(\cdot)$ denotes the resolution-first clipping strategy, A_i^{res} and A_i^{eff} represent the resolution and efficiency advantages of the i-th rollout in $\mathcal{G}$, respectively. In the implementation, we derive A_i^{res} using the standard group-relative advantage estimator in GRPO (Shao et al., 2024), with the binary resolution outcome 0 or 1 as the reward. More specifically, we adopt an efficiency gating mechanism to balance the trade-off between resolution performance and token efficiency, *i.e.*,

$$w = \lambda \,\text{clip}\left(\frac{p-\tau}{1-\tau}, 0, 1\right), \quad p = \frac{1}{|\mathcal{G}|}\sum_{i\in\mathcal{G}} y_i. \tag{5}$$

where λ controls the strength of efficiency optimization, p indicates the average resolution rate within the group $\mathcal{G}$, and τ denotes the threshold that determines whether efficiency optimization is activated. Such a mechanism encourages coding agents to learn efficient reasoning patterns only when they demonstrate sufficient capability to resolve the given task, thereby prioritizing task resolution before optimizing token efficiency.

Although the efficiency gating mechanism poses constraints on efficiency optimization, the efficiency signal might override the resolution signal, causing unresolved trajectories to receive higher advantages than resolved ones. To avoid this, we employ the resolution-first clipping strategy as follows,

$$\Gamma(A_i) = \begin{cases} \max(A_i, 0), & y_i = 1 \\ \min(A_i, 0), & y_i = 0 \end{cases}. \tag{6}$$

Thanks to the resolution-first clipping strategy, advantages are non-negative for resolved trajectories and non-positive for unresolved ones, thereby further preserving the priority of task resolution during policy optimization.

3.3 MULTI-GRANULARITY EFFICIENCY CREDIT ASSIGNMENT

To prevent inefficient behaviors from incurring substantial token overhead, we introduce a multi-granularity credit assignment mechanism, which assigns efficiency credit at both the coarse-grained trajectory level and the fine-grained turn level. Specifically, the efficiency advantage is derived as follows,

$$A_{i,j}^{\text{eff}} = A_i^{\text{coarse}} + A_{i,j}^{\text{fine}}, \tag{7}$$

where A_i^{coarse} denotes the coarse-grained advantage of the i-th rollout, $A_{i,j}^{\text{fine}}$ indicates the fine-grained advantage of j-th turn in the corresponding rollout. In the following, we will elaborate on the two advantages in detail.

3.3.1 COARSE-GRAINED TRAJECTORY ADVANTAGE

To facilitate the learning of efficient reasoning patterns, a straightforward approach is to penalize token-intensive trajectories while encouraging concise ones. However, recent studies (Fan et al., 2025; Majgaonkar et al., 2025) have demonstrated that unresolved trajectories typically consume more tokens than resolved ones. As a result, simply rewarding trajectories with lower token usage might conflate token efficiency with resolution rate rather than encouraging more efficient reasoning patterns.

To remedy this, we adopt the resolution-calibrated credit assignment strategy as follows,

$$A_i^{\text{coarse}} = \frac{u_i - \bar{u}}{\sigma_u},$$
$$u_i = 1 - \tanh\left(\max\left(\frac{c_i}{B^{y_i}} - 1,\, 0\right)\right), \qquad B^{y_i} = \text{Median}\left\{c_k \mid k \in \mathcal{G},\, y_k = y_i\right\}, \tag{8}$$

where u_i denotes the efficiency reward for the i-th rollout, $\text{Median}\{\cdot\}$ indicates the median token usage among rollouts in $\mathcal{G}$ with the same resolution outcome, and $\bar{u}$ and σ_u represent the mean and standard deviation of the efficiency rewards within $\mathcal{G}$, respectively. Intuitively, efficiency rewards are calibrated within each outcome subset, thus avoiding the conflation of token efficiency with resolution performance. After that, the calibrated rewards are standardized to obtain trajectory-level advantages, facilitating the learning of efficient reasoning patterns under the same resolution outcome.

3.3.2 Fine-grained Turn Advantage

Although the proposed coarse-grained advantage could reduce token usage to some extent, the trajectory-level signal treats all interaction turns equally, which overlooks that the unproductive turns are the primary source of substantial token overhead.

As discussed in the Introduction, we reveal that low-entropy turns are more likely to exhibit unproductive behaviors. Motivated by this, we propose the following entropy-aware credit assignment mechanism to avoid inefficient behaviors, *i.e.*,

$$A_{i,j}^{\text{fine}} = \begin{cases} -\tanh\left(1 - \dfrac{h_{i,j}}{q}\right), & h_{i,j} < q \\ 0, & h_{i,j} \geq q \end{cases}, \tag{9}$$
$$q = \text{Percentile}_{0.2}\left(\left\{h_{i,j} \mid i \in \mathcal{G},\, j \in \{1, \ldots, n_i\}\right\}\right),$$

where $h_{i,j}$ denotes the average token entropy at the j-th turn of the i-th rollout. Following Wang et al. (2026a), $\text{Percentile}_{0.2}(\cdot)$ returns the entropy threshold corresponding to the lowest 0.2 fraction of turns within the group. Such a design would penalize low-entropy turns potentially associated with inefficient behaviors, thereby reducing the token overhead incurred by unproductive interactions.

4 Experiments

In this section, we conduct extensive experiments on widely-used SWE benchmarks to verify the effectiveness of the proposed HERO. Due to space limitations, we present more experiments in Appendix B.

4.1 Experiment Configurations

Benchmarks. To comprehensively evaluate the effectiveness of HERO, we evaluate our method on the most widely-used benchmarks in SWE scenarios, *i.e.*, SWE-bench Verified and SWE-bench Multilingual (Jimenez et al., 2024). Specifically, SWE-bench Verified contains 500 high-quality issue-solving tasks constructed from real-world GitHub issues, covering bug fixing, feature implementation, code refactoring, and so on. While SWE-bench Multilingual extends the evaluation beyond Python, comprising 300 SWE tasks across nine programming languages.

Implementation Details. We evaluate our proposed HERO using the widely-used SOTA model Qwen3.5 (Yang et al., 2025) and conduct comprehensive experiments on three variants with different model sizes, *i.e.*, Qwen3.5-4B, Qwen3.5-9B, and Qwen3.5-35B-A3B. For a fair comparison, we adopt the most widely-acknowledged scaffold, *i.e.*, Claude Code, as the unified framework for evaluating all coding agents and RL methods. As for comparisons between RL methods, we use the same training set of 640 multilingual SWE tasks sampled from SWE-Gym (Pan et al., 2024), Multi-SWE-bench (Zan et al., 2026), and SWE-rebench (Badertdinov et al., 2026). See Appendix A.1 for more details. Regarding hyperparameter settings, the trade-off coefficient λ and threshold τ in Eq. 5 are set to 0.3 and 0.6, respectively. We repeat all experiments three times with different random seeds and report the mean results.

Table 1: Comparisons with state-of-the-art methods on SWE-bench Verified benchmark. Token usage is reported in millions (M).

Method	Resolve Rate (%)	Token Usage (M)	TRS
Open-Source Coding Agents			
Gemma-4-26B-A4B	35.4	2.3	19.5
Qwen3-Coder-30B-A3B-Instruct	43.2	2.4	23.4
GLM-4.7-Flash-30B	42.6	4.6	18.0
KAT-Dev-32B	48.4	2.4	26.2
Reinforcement Learning			
Qwen3.5-4B	32.8	4.5	14.0
+ GRPO	37.4	6.8	13.4
+ Our	40.0	4.5	17.1
Qwen3.5-9B	45.0	4.5	19.2
+ GRPO	47.0	5.9	17.9
+ Our	49.2	3.5	23.2
Qwen3.5-35B-A3B	57.2	2.9	29.0
+ GRPO	59.2	5.2	23.8
+ Our	60.6	2.7	31.5

Table 2: Comparisons with state-of-the-art methods on SWE-bench Multilingual benchmark.

Method	Resolve Rate (%)	Token Usage (M)	TRS
Open-Source Coding Agents			
Gemma-4-26B-A4B	27.3	3.5	12.9
Qwen3-Coder-30B-A3B-Instruct	20.6	2.1	11.7
GLM-4.7-Flash-30B	25.3	4.0	11.3
KAT-Dev-32B	27.6	2.3	15.2
Reinforcement Learning			
Qwen3.5-4B	20.0	4.0	8.9
+ GRPO	22.0	7.6	7.5
+ Our	24.3	4.5	10.4
Qwen3.5-9B	29.3	4.3	12.7
+ GRPO	31.0	6.0	11.7
+ Our	34.0	4.1	15.1
Qwen3.5-35B-A3B	40.3	3.2	19.7
+ GRPO	43.0	5.4	17.0
+ Our	45.0	2.9	22.8

4.2 Comparisons with State-of-the-Art Methods

We compare our method with the widely-used reinforcement learning method, *i.e.*, GRPO (Shao et al., 2024), using Qwen3.5 backbones of various model sizes. For more results on state-of-the-art token-efficient reinforcement learning methods, please refer to Appendix B.1. Besides, we compare against four state-of-the-art coding agents, i.e., Gemma-4-26B-A4B (Team et al., 2026), Qwen3-

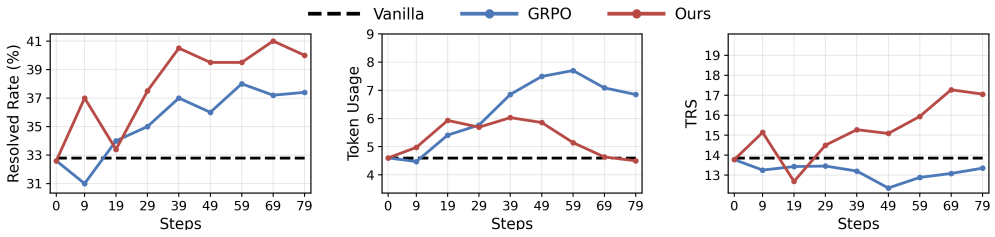


Figure 3: Dynamics of the training process.

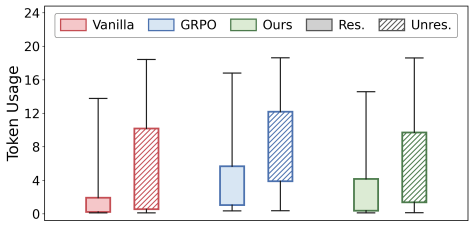


Figure 4: Token usage distributions of resolved and unresolved trajectories.

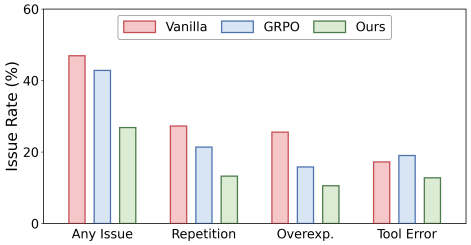


Figure 5: Issue rate of unproductive behaviors at turn level.

Coder-30B-A3B-Instruct (Yang et al., 2025), GLM-4.7-Flash-30B (Zeng et al., 2026), and KAT-Dev-32B (Zhan et al., 2025).

From the results in Table 1-2, one could have the following observations and conclusions: i) state-of-the-art coding agents with strong resolution performance do not necessarily exhibit high token efficiency; ii) GRPO consistently improves resolution rates at the cost of substantially increased token usage, suggesting that GRPO could enhance issue-solving capabilities without necessarily encouraging efficient reasoning patterns; iii) in contrast, the proposed HERO prioritizes task resolution over token efficiency while reducing the token overhead incurred by unproductive interactions, achieving desirable resolution rates and token efficiency; iv) thanks to the proposed HERO, even small models such as Qwen3.5-4B and Qwen3.5-9B achieve competitive resolution performance against larger open-source SOTA coding agents.

4.3 Ablation and Analytic Study

In this section, we carry out a series of ablation studies and analytic experiments to investigate the effectiveness of HERO. Unless otherwise stated, experiments are conducted using Qwen3.5-4B on the SWE-bench Verified benchmark.

Analytic Study on Training Dynamics. To investigate how HERO learns efficient reasoning patterns and improves resolution performance, we carry out an analytic study on the dynamics throughout the training process, including resolved rate, token usage, and TRS. As demonstrated in Fig. 3, HERO outperforms GRPO across the three metrics as training progresses, with more pronounced gains in later stages. Besides, we observe that token usage under HERO first rises until step 40 and then declines, whereas the resolution rate reaches a high level by this stage and remains consistently high thereafter. In other words, such a phenomenon suggests that HERO first develops issue-solving capabilities in the early stages and subsequently reduces token usage while preserving resolution performance in the later stage, thereby achieving a favorable trade-off between the two objectives.

Analytic Study on Token Usage. To verify the effectiveness of HERO in improving token efficiency, we compare token usage distributions for resolved and unresolved trajectories under GRPO and the proposed HERO. From the results in Fig. 4, HERO achieves lower token usage for both resolved and unresolved trajectories compared with GRPO, suggesting that our method could learn efficient reasoning patterns and thus achieve resolution with fewer tokens.

Analytic Study on Unproductive Turns. As pointed out in Introduction, low-entropy turns are more prone to inefficient behaviors. To verify the effectiveness of HERO in suppressing unproduc-

Table 3: Ablation studies on the designs in HERO, where "w/o HPO" denotes removing the hierarchical policy optimization.

Variants	Resolve Rate	Token Usage	TRS
GRPO	37.4	6.8	13.4
w/o HPO	29.2	2.1	16.6
w/o Coarse	39.0	6.0	14.7
w/o Fine	37.8	4.9	15.6
Default	40.0	4.5	17.1

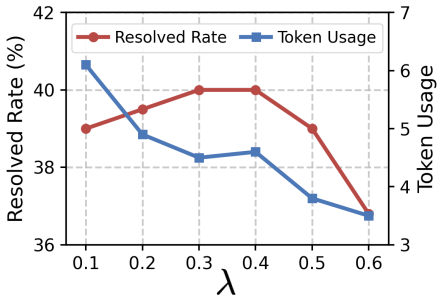


Figure 6: Parameter analysis of efficiency coefficient λ in Eq. 5.

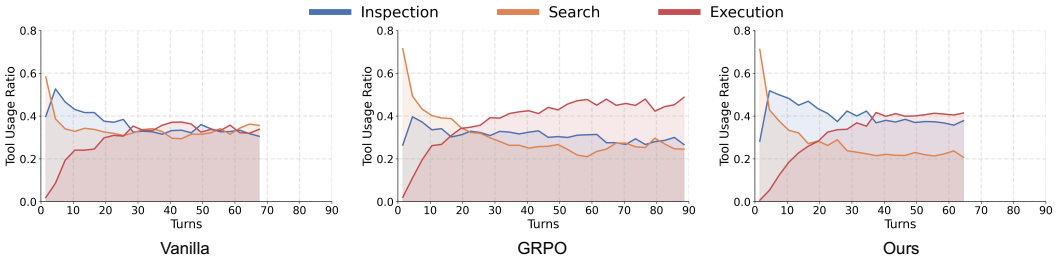


Figure 7: Tool usage across various interaction turns.

tive turns, we analyze the frequencies of unproductive behaviors across interaction turns. As shown in Fig. 5, HERO achieves lower rates of repeated tool calls, overexploration, and tool errors than both the base model and GRPO, suggesting that penalizing low-entropy turns would discourage unproductive behaviors.

Ablation Studies. To verify the importance of each design, we investigate the variants of HERO in Table 3, from which one could draw the following conclusions. On the one hand, removing hierarchical policy optimization would reduce token usage but degrade resolution performance, highlighting its role in preserving task resolution during efficiency optimization. On the other hand, both the coarse-grained and fine-grained efficiency advantages contribute to achieving lower token usage. Besides, we carry out parameter analysis of the efficiency coefficient λ in Eq. 5. As depicted in Fig. 6, HERO achieves stable performance improvement with low token consumption when $\lambda \in [0.2, 0.4]$. See Appendix B.8-B.9 for ablation studies on more parameters.

Analytic Study on Reasoning Patterns. To further verify that HERO learns both effective and efficient reasoning patterns, we conduct an analytical study on the tool usage across interaction turns. From the results in Fig. 7, we observe that HERO shifts from inspection toward execution in later turns, mirroring the reasoning progression of GRPO while requiring fewer tokens. In other words, HERO internalizes more efficient reasoning patterns into the agent policy without undermining the necessary progression from code exploration to solution generation.

5 Conclusion

In this paper, we study a highly-practical yet under-explored problem, *i.e.*, training token-efficient coding agents while preserving resolution performance. To solve this problem, we propose a novel RL framework, *i.e.*, HERO, which achieves efficient reasoning patterns through multi-granularity credit assignment and prioritizes task resolution over token efficiency through hierarchical policy optimization. Extensive experiments on SWE-bench series demonstrate that HERO significantly outperforms existing methods and achieves a favorable trade-off between resolution rate and token efficiency. In the future, we plan to extend HERO to more complex long-horizon agentic tasks, such as scientific data analysis and engineering design.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods. We have not used generative AI tools for generating synthetic data sets, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, assisting with translation, cleaning and reformatting datasets, or interpreting results, and providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for editing a research paper to improve readability. We have reviewed all AI-assisted work. Specifically, we check AI-assisted code for correctness and AI-assisted text for technical accuracy and consistency with our intended meaning. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Daman Arora and Andrea Zanette. Training language models to reason efficiently. In *NeurIPS*, 2026.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *NeurIPS*, 2026.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, 2024.

Zhiyu Fan, Kirill Vasilevski, Dayi Lin, Boyuan Chen, Yihao Chen, Zhiqing Zhong, Jie M Zhang, Pinjia He, and Ahmed E Hassan. Swe-effi: Re-evaluating software ai agent system effectiveness under resource constraints. *arXiv preprint arXiv:2509.09853*, 2025.

Pengfei Gao and Chao Peng. More with less: An empirical study of turn-control strategies for efficient coding agents. *arXiv preprint arXiv:2510.16786*, 2025.

Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 2025.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Dong Huang, Jie M Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*, 2023.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *ICLR*, 2024.

Haolin Jin, Linghan Huang, Haipeng Cai, Jun Yan, Bo Li, and Huaming Chen. From llms to llm-based agents for software engineering: A survey of current, challenges and future. *arXiv preprint arXiv:2408.02479*, 2024.

Haobin Li, Yutong Yang, Yijie Lin, Xiang Dai, Mouxing Yang, and Xi Peng. Reliable thinking with images. *arXiv preprint arXiv:2602.12916*, 2026.

Tobias Lindenbauer, Igor Slinko, Ludwig Felder, Egor Bogomolov, and Yaroslav Zharov. The complexity trap: Simple observation masking is as efficient as llm summarization for agent context management. *arXiv preprint arXiv:2508.21433*, 2025.

Jiacheng Liu, Xiaohan Zhao, Xinyi Shang, and Zhiqiang Shen. Dive into claude code: The design space of today's and future ai agent systems. *arXiv preprint arXiv:2604.14228*, 2026a.

Wei Liu, Ruochen Zhou, Yiyun Deng, Yuzhen Huang, Junteng Liu, Yuntian Deng, Yizhe Zhang, and Junxian He. Learn to reason efficiently with adaptive length-based reward shaping. In *ICLR*, 2026b.

Abhilasha Lodha, Mahsa Pahlavikhah Varnosfaderani, Abir Chakraborty, and Abhinav Mithal. Less context, better agents: Efficient context engineering for long-horizon tool-using llm agents. *arXiv preprint arXiv:2606.10209*, 2026.

Oorja Majgaonkar, Zhiwei Fei, Xiang Li, Federica Sarro, and He Ye. Understanding code agent behaviour: An empirical study of success and failure trajectories. *arXiv preprint arXiv:2511.00197*, 2025.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. *arXiv preprint arXiv:2402.06196*, 2024.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Vaishnavi Shrivastava, Ahmed H Awadallah, Vidhisha Balachandran, Shivam Garg, Harkirat Behl, and Dimitris Papailiopoulos. Sample more to think less: Group filtered policy optimization for concise reasoning. In *ICLR*, 2026.

Huatong Song, Lisheng Huang, Shuang Sun, Jinhao Jiang, Ran Le, Daixuan Cheng, Guoxin Chen, Yiwen Hu, Zongchao Chen, Yiming Jia, et al. Swe-master: Unleashing the potential of software engineering agents via post-training. *arXiv preprint arXiv:2602.03411*, 2026.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Otc: Optimal tool calls via reinforcement learning. *arXiv preprint arXiv:2504.14870*, 2025a.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xiong-Hui Chen, Jianxin Yang, Zhenru Zhang, et al. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. In *NeurIPS*, 2026a.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *ICLR*, 2025b.

Yuhang Wang, Yuling Shi, Mo Yang, Rongrui Zhang, Shilin He, Heng Lian, Yuting Chen, Siyu Ye, Kai Cai, and Xiaodong Gu. Swe-pruner: Self-adaptive context pruning for coding agents. *arXiv preprint arXiv:2601.16746*, 2026b.

Yuhang Wang, Yuling Shi, Shaoqiu Zhang, Jialiang Liang, Shilin He, Siyu Ye, Yuting Chen, Kai Cai, and Xiaodong Gu. Swe-pruner pro: The coder llm already knows what to prune. *arXiv preprint arXiv:2607.18213*, 2026c.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

Zequn Xie, Junjie Wang, Dan Yang, Jie Feng, Yue Shen, Jian Wang, and Jinjie Gu. Slim-searcher: Training efficiency-aware web agents via adaptive reward gating. *arXiv preprint arXiv:2606.07074*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *NeurIPS*, 2024.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Li Aoyan, Lu Chen, Xiaojian Zhong, et al. Multi-swe-bench: A multilingual benchmark for issue resolving. In *NeurIPS*, 2026.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Zizheng Zhan, Ken Deng, Jinghui Wang, Xiaojiang Zhang, Huaixi Tang, Minglei Zhang, Zhiyi Lai, Haoyang Huang, Wen Xiang, Kun Wu, et al. Kat-coder technical report. *arXiv preprint arXiv:2510.18779*, 2025.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *ACL*, 2024.

APPENDIX

A More Implementation Details **14**

A.1 Training Data . . . 14

A.2 More Implementation Settings . . . 14

A.3 Entropy Estimation . . . 14

A.4 Identification of Unproductive Behaviors . . . 14

B Additional Experimental Results **16**

B.1 Comparisons with Token-Efficient Reinforcement Learning Methods . . . 16

B.2 Comparisons with Inference-Time Methods . . . 16

B.3 Analytic Study on Prefix-Cache Reuse and Inference Cost . . . 17

B.4 Ablation Studies on SWE-bench Multilingual . . . 17

B.5 Analytic Study on Token Usage Breakdown . . . 18

B.6 Analytic Study on Scaffold Generalization . . . 18

B.7 More Detailed Ablation Studies . . . 19

B.8 Sensitivity to the Capability Threshold . . . 19

B.9 Sensitivity to the Entropy Percentile . . . 19

B.10 Analytic Study on Fine-grained Turn Advantage . . . 20

A More Implementation Details

In this section, we provide more implementation details about the training data, implementation settings, entropy estimation, and identification of unproductive behaviors.

A.1 Training Data

As mentioned in Section 4, we employ 640 multilingual SWE tasks for training. Specifically, the training set contains 352 Python tasks, 38 Java tasks, and 50 tasks each in C/C++, JavaScript/TypeScript, PHP, Go, and Rust. Such a collection covers SWE tasks across multiple programming languages, endowing the training of coding agents with multilingual reasoning capabilities.

A.2 More Implementation Settings

We provide additional implementation details for training and evaluation. Specifically, the default tool configuration disables WebFetch, WebSearch, and Agent, while retaining all other scaffold-provided tools. For training, each training rollout batch contains 8 tasks with 16 sampled trajectories per task, yielding 128 trajectories in total. Moreover, in Eq. 4, the derived turn-level advantage is assigned to every generated token within the turn. Note that, given the stronger issue-solving capability of Qwen3.5-35B-A3B, the trade-off coefficient λ and threshold τ in Eq. 5 are set to 0.1 and 0.8, respectively. For evaluation, we use the checkpoint from the final training iteration. Together, the default settings are summarized in Table 4.

Table 4: Default implementation settings.

Category	Parameter	Setting
Agent Scaffold	Claude Code version	2.1.111
Training	Tasks per rollout batch	8
	Rollout group size	16
	Trajectories per rollout batch	128
Evaluation	Sampling temperature	0.6
	Top-p	0.95
	Maximum interaction turns	200
	Context length	131,072 tokens

A.3 Entropy Estimation

We provide additional details on the estimation of turn-level entropy. Following Holtzman et al. (2019); Li et al. (2026), we estimate token-level entropy using the top-10 token probabilities and derive turn-level entropy as

$$h_{i,j} = \frac{1}{|\mathcal{T}_{i,j}|} \sum_{k \in \mathcal{T}_{i,j}} H_k, \tag{10}$$

where $\mathcal{T}_{i,j}$ denotes the set of generated tokens in the j-th turn of rollout i, with H_k denotes the entropy of the k-th generated token.

A.4 Identification of Unproductive Behaviors

In the manuscript, we analyze the association between turn-level entropy and unproductive behaviors in Fig. 1. Here, we provide the definition for repeated tool calls, excessive exploration, and erroneous commands in Table 5. Note that each behavior is counted at most once per turn.

Table 5: Identification rules for unproductive behaviors.

Behavior	Identification Rule
Repetition	Repeats the same sequence of tool calls and arguments as the preceding turn.
Overexploration	Uses only `Glob`, `Grep`, `Read`, or `NotebookRead` for at least 15 consecutive turns.
Tool Error	Tool responses contain explicit tool-error markers, excluding failed tests or nonzero exit codes alone.
Any Issue	Exhibits at least one of the above behaviors.

B Additional Experimental Results

In this section, we present more experimental results of the proposed HERO. Unless otherwise specified, all experiments are conducted using Qwen3.5-4B.

B.1 Comparisons with Token-Efficient Reinforcement Learning Methods

As discussed in Introduction, existing token-efficient RL methods are specifically designed for non-interactive reasoning or simple agentic tasks, which struggle to achieve a favorable trade-off between resolution rate and token efficiency in complex SWE scenarios. We compare HERO with four state-of-the-art token-efficient RL methods, *i.e.*, Training Efficient (Arora & Zanette, 2026), LASER (Liu et al., 2026b), OTC-GRPO (Wang et al., 2025a), and SlimSearcher (Xie et al., 2026), on SWE-bench Verified and SWE-bench Multilingual. For fair comparison, we adopt the same training and evaluation settings across methods, except for method-specific configurations. Specifically, for Training Efficient, OTC-GRPO, and SlimSearcher, we follow the configurations reported in their respective papers. While for LASER, we use the fixed-threshold variant with a threshold of 8,192 total output tokens across interaction turns, estimated based on the generation lengths typical of SWE tasks.

From the results in Table 6, one could have the following observations and conclusions. On the one hand, HERO achieves higher resolution rates with lower token usage than Training Efficient and SlimSearcher on both benchmarks. On the other hand, although LASER and OTC-GRPO consume fewer tokens, these token savings come at the cost of degrading resolution rates, highlighting the difficulty of balancing resolution performance and token efficiency for existing token-efficient RL methods. Overall, HERO significantly outperforms existing token-efficient RL methods in the TRS metric, demonstrating that HERO achieves a more favorable trade-off between the two objectives.

Table 6: Comparisons with token-efficient RL methods on SWE-bench Verified and SWE-bench Multilingual.

	SWE-bench Verified			SWE-bench Multilingual		
Method	Resolved Rate (%)	Tokens (M)	TRS	Resolved Rate (%)	Tokens (M)	TRS
Base	32.8	4.5	14.0	20.0	4.0	8.9
GRPO	37.4	6.8	13.4	22.0	7.6	7.5
Training Efficient	37.4	6.4	13.7	21.3	6.8	7.6
LASER	33.2	3.4	15.8	21.7	4.8	9.0
OTC-GRPO	32.4	4.0	14.5	17.0	3.2	8.3
SlimSearcher	37.6	6.3	13.9	21.0	6.4	7.7
Ours	40.0	4.5	17.1	24.3	4.5	10.4

B.2 Comparisons with Inference-Time Methods

In the manuscript, we argue that existing inference-time approaches struggle to balance the trade-off between resolution performance and token efficiency. To verify the claim, we conduct additional experiments on four representative inference-time approaches, *i.e.*, SWE-Pruner (Wang et al., 2026b), Pruning + Summ. (Lodha et al., 2026), Observation Masking (Lindenbauer et al., 2025), and Turn Limit (Gao & Peng, 2025). Specifically, we apply inference-time approaches to both Qwen3.5-4B and its HERO-trained counterpart, termed HERO-4B.

From the results in Table 7, one could have the following observations and conclusions: i) most inference-time approaches reduce token usage without improving resolution performance over the base model, highlighting the difficulty of balancing resolution performance and token efficiency; ii) HERO-4B achieves higher TRS scores than inference-time approaches applied to the base model, suggesting that our training-time designs achieve a more favorable trade-off between resolution performance and token efficiency; iii) thanks to the training-time designs, HERO-4B could still benefit from inference-time approaches, suggesting that the proposed HERO could complement inference-time approaches.

Table 7: Comparisons with inference-time approaches on SWE-bench Verified and SWE-bench Multilingual.

	SWE-bench Verified			SWE-bench Multilingual		
Variants	Resolved Rate	Tokens	TRS	Resolved Rate	Tokens	TRS
Qwen3.5-4B	32.8	4.5	14.0	20.0	4.0	8.9
+ SWE-Pruner	33.0	4.3	14.3	19.7	3.8	9.0
+ Pruning + Summ.	31.4	3.2	15.3	18.7	3.3	9.0
+ Observation Masking	30.0	3.8	13.7	18.0	3.5	8.5
+ Turn Limit	29.0	3.0	14.5	16.7	3.1	8.2
HERO-4B	40.0	4.5	17.1	24.3	4.5	10.4
+ SWE-Pruner	40.4	4.3	17.5	23.7	4.4	10.2
+ Pruning + Summ.	36.4	3.4	17.4	21.7	3.6	10.1
+ Observation Masking	39.4	3.8	18.0	23.4	3.8	10.7
+ Turn Limit	36.0	3.1	17.8	21.0	3.1	10.4

B.3 Analytic Study on Prefix-Cache Reuse and Inference Cost

To further investigate efficiency in real-world applications, we conduct an analytic study on prefix-cache reuse and relative inference cost on SWE-bench Verified. Following Lindenbauer et al. (2025), we estimate inference cost post hoc from recorded token usage. Specifically, we derive the token-weighted cache-hit rate and per-instance cost as

$$\begin{aligned} \mathrm{HitRate} &= 100\frac{\sum_i t_i^{\mathrm{cache}}}{\sum_i t_i^{\mathrm{in}}}, \\ \mathrm{Cost}_i &= \left(t_i^{\mathrm{in}} - t_i^{\mathrm{cache}}\right) + 0.1t_i^{\mathrm{cache}} + 4t_i^{\mathrm{out}}, \end{aligned} \tag{11}$$

where t_i^{in}, t_i^{cache}, and t_i^{out} denote the cumulative input, cached input, and output token counts for instance i. We employ fixed cost weights of 1, 0.1, and 4 to uncached input, cached input, and output tokens, respectively. Relative inference cost is normalized by the average per-instance cost of the corresponding base model at each model size.

From the results in Table 8, one could have the following conclusions. On the one hand, although context manipulation methods, *e.g.*, Pruning + Summ. and Observation Masking, could reduce token usage, these methods would undermine prefix-cache reuse, resulting in higher relative inference costs. On the other hand, interaction limits reduce both token usage and relative inference cost at the expense of a serious degradation in resolution performance. In contrast, HERO achieves competitive prefix-cache reuse and relative inference cost, supporting its effectiveness in practical applications.

Table 8: Analytic study on prefix-cache reuse and relative inference cost on SWE-bench Verified.

Variants	Resolved Rate (%)	Cache Hit (%)	Relative Cost
Qwen3.5-4B	32.8	92.4	1.0
+ SWE-Pruner	33.0	86.3	1.4
+ Pruning + Summ.	31.4	71.9	1.3
+ Observation Masking	30.0	80.3	1.3
+ Turn Limit	29.0	89.2	0.7
HERO-4B	40.0	91.9	1.0

B.4 Ablation Studies on SWE-bench Multilingual

In the manuscript, we have conducted ablation studies on SWE-bench Verified. Here, we extend the ablation studies to more complex multilingual SWE tasks, *i.e.*, SWE-bench Multilingual. As shown in Table 9, removing hierarchical policy optimization (HPO) reduces token usage at the cost of

degrading resolution performance, while either coarse-grained or fine-grained efficiency advantages contribute to reducing token consumption and improving resolution performance.

Table 9: Ablation studies of HERO on SWE-bench Multilingual.

Variants	Resolved Rate	Tokens	TRS
GRPO	22.0	7.6	7.5
w/o HPO	17.0	2.3	9.4
w/o Coarse	23.7	7.0	8.4
w/o Fine	22.3	5.5	8.7
Default	24.3	4.5	10.4

B.5 Analytic Study on Token Usage Breakdown

We conduct more in-depth analysis to investigate the overhead of multi-turn interactions. From the results in Tables 10 and 11, input tokens account for more than 99% of total consumption, highlighting the significant overhead of repeatedly processing accumulated context. Compared with GRPO, HERO achieves higher resolution rates with fewer interaction turns and lower token consumption on both benchmarks.

Table 10: Token usage and interaction turns on SWE-bench Verified. "Input Ratio" denotes the proportion of input tokens in total token consumption.

Method	**Resolved Rate**	**Total Tokens**	**Input Ratio**	**Avg. Turns**
Qwen3.5-4B	32.8	4.5	99.6	68.7
+ GRPO	37.4	6.8	99.6	94.8
+ Ours	40.0	4.5	99.4	65.6

Table 11: Token usage and interaction turns on SWE-bench Multilingual.

Method	**Resolved Rate**	**Total Tokens**	**Input Ratio**	**Avg. Turns**
Qwen3.5-4B	20.0	4.0	99.5	64.3
+ GRPO	22.0	7.6	99.6	106.5
+ Ours	24.3	4.5	99.3	69.6

B.6 Analytic Study on Scaffold Generalization

In the manuscript, we have evaluated the proposed HERO under the unified Claude Code scaffold for fairness. Here, we conduct additional experiments using widely-used mini-SWE-agent scaffold on SWE-bench Verified. As shown in Table 12, even training on the Claude Code scaffold, HERO achieves higher resolution rates with lower token consumption than GRPO under both scaffolds, indicating HERO learns efficient reasoning patterns that generalize across scaffolds.

Table 12: Performance comparisons under different agent scaffolds on SWE-bench Verified.

	Claude Code			mini-SWE-agent		
Variants	Resolved Rate	Tokens	TRS	Resolved Rate	Tokens	TRS
Qwen3.5-4B	32.8	4.5	14.0	39.5	4.0	17.7
+ GRPO	37.4	6.8	13.4	41.2	5.2	16.5
+ Our	40.0	4.5	17.1	43.0	4.0	19.2

B.7 More Detailed Ablation Studies

To further verify the effectiveness of individual designs, we carry out ablation studies on the efficiency gating and resolution-first clipping in hierarchical policy optimization (HPO) and outcome conditioning in coarse-grained efficiency advantage. As demonstrated in Table 13, one could have the following conclusions: i) either efficiency gating or resolution-first clipping could help preserve resolution performance, which verifies the effectiveness of hierarchical policy optimization in prioritizing task resolution over token efficiency; ii) resolution-calibrated credit assignment could facilitate the learning of efficient reasoning patterns without conflating token efficiency with resolution performance, thus achieving a better trade-off between resolution rate and token efficiency.

Table 13: More detailed ablation studies on SWE-bench Verified. "w/o Calibration" denotes using shared threshold across all trajectories within the rollout group.

Component	Variants	Resolved Rate	Tokens	TRS
HPO	w/o Efficiency Gating	36.0	3.8	16.4
	w/o Clipping	37.8	4.1	16.7
	Default	40.0	4.5	17.1
Coarse	w/o Calibration	38.2	5.9	14.5
	Default	40.0	4.5	17.1

B.8 Sensitivity to the Capability Threshold

We further investigate the effect of the capability threshold τ in Eq. 5. As shown in Fig. 8, HERO achieves a desirable trade-off between resolution performance and token efficiency when $\tau \in [0.5, 0.6]$.

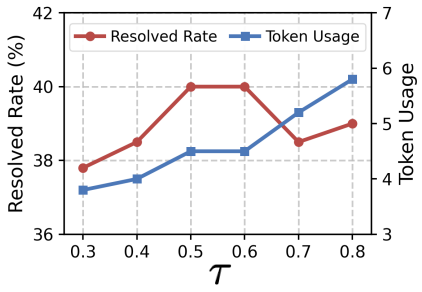


Figure 8: Parameter analysis of the capability threshold τ in Eq. 5.

B.9 Sensitivity to the Entropy Percentile

We conduct additional parameter analysis of the entropy percentile in Eq. 9. As shown in Fig. 9, a larger percentile generally reduces token usage but also degrades resolution performance, suggesting that penalizing too many turns might suppress necessary reasoning.

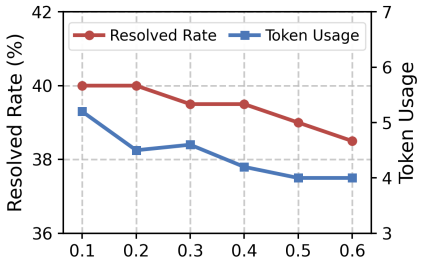


Figure 9: Parameter analysis of the entropy percentile in Eq. 9.

B.10 Analytic Study on Fine-grained Turn Advantage

To further investigate the effectiveness of the fine-grained turn advantage, we compare two alternative penalty strategies on SWE-bench Verified: i) Random: randomly selecting 20% of turns for penalization; ii) High Entropy: selecting the 20% of turns with the highest entropy for penalization. As demonstrated in Table 14, the default strategy of penalizing low-entropy turns achieves a higher resolution rate with lower token consumption than both alternatives, suggesting that penalizing low-entropy turns contributes to learning efficient reasoning patterns.

Table 14: Analytic study of turn-level penalty strategies on SWE-bench Verified.

Variants	Resolved Rate (%)	Tokens (M)	TRS
Random	36.8	5.4	14.5
High Entropy	36.0	6.0	13.6
Default	40.0	4.5	17.1