Hierarchical Reward Modeling for Fault Localization in Large Code Repositories

Jiwei Zhang[1], Jianxun Lian[2,*], Haiming Qin[1], Mingyang Zhou[1,*], KeZhong Lu[1], Rui Mao[1], Hao Liao[1,*]

[1]College of Computer Science and Software Engineering, Shenzhen University, China

[2]Microsoft Research Asia

2350273005@email.szu.edu.cn jianxun.lian@outlook.com

22453103002@mails.szu.edu.cn {zmy, kzlu, mao, haoliao}@szu.edu.cn

Abstract

Large Language Models (LLMs) exhibit significant potential in complex software engineering tasks, however, their fault localization capabilities within repository are constrained by inherent limitations in max context length. Although Test-Time Scaling (TTS) can generate multiple candidate solutions, traditional selection strategies often fail to identify the optimal one. To solve this problem, we introduces **Hi**erarchical **Lo**calization **R**eward **M**odel (HiLoRM), which specifically designed to evaluate and select the most accurate fault localization candidates (at file, function, and line levels) from the multiple sampled outputs of LLMs, thereby enhancing localization accuracy. Furthermore, we constructed the HiFL-44k dataset, comprising approximately 44,000 fault localization instances, to train HiLoRM. Experimental results demonstrate that on the SWE-Bench-Lite dataset, HiLoRM improves the final line-level localization recall by 12% compared to a baseline model that does not use a reward model. Concurrently, HiLoRM exhibits a strong capability to evaluate predictions from larger LLMs (e.g., 32B parameters) and demonstrates transferability and generalization potential when applied to other fault localization methods. This work provides an effective methodology and an accessible model to significantly improve the accuracy and reliability of LLMs for repository-level fault localization. Our codes and dataset are available at https://github.com/SZU-ZJW/HiFL-Method

1 Introduction

Large Language Models (LLMs) have demonstrated potential in tackling complex software engineering tasks within intricate code repositories, such as resolving GitHub issues. However, their performance on benchmarks like SWE-Bench remains limited (Jimenez et al., 2024; Yang et al., 2024c), with even state-of-the-art open-source LLMs often struggling to exceed a 50% resolution rate. In-depth analysis reveals that fault localization—precisely identifying code locations needing modification—is a critical bottleneck hindering LLM performance (Qin et al., 2024). The inherent input context length limitations of current LLMs prevent processing entire large repositories, making precise identification of relevant code segments challenging. Effective fault localization is thus a fundamental prerequisite for successful code repair (Wang et al., 2025).

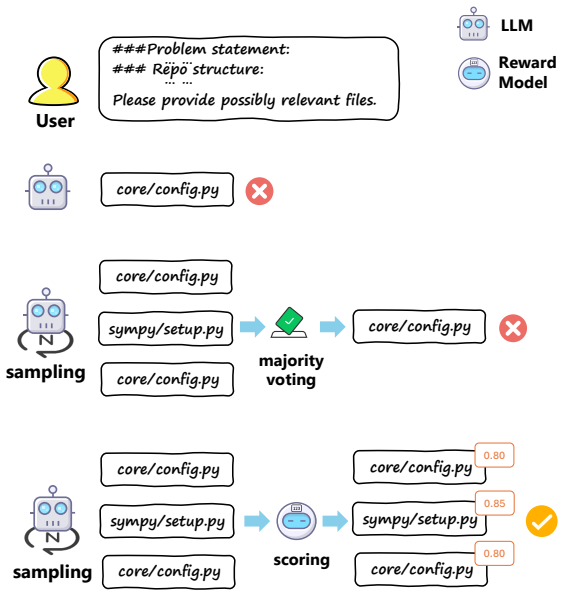


Figure 1: Comparative example of test-time scalling and reward model selection strategy.

To address LLM performance limitations in code repair, researchers have explored strategies like more fine-grained agent methods (Yang et al., 2024c; Tao et al., 2024), Retrieval-Augmented Generation (RAG) for enhanced repository understanding (Wadhwa et al., 2024), and fine-tuning LLMs for specific domains (Liu et al., 2024). Test Time

* Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 17782–17796

Scaling (TTS), a technique improving model output quality by allocating additional computational resources during testing without model retraining, has also shown promise, with improved performance in models like OpenAI's o1 and DeepSeek-R1 (Jaech et al., 2024; Guo et al., 2025). TTS typically generates multiple candidate solutions, with one subsequently selected. However, conventional TTS selection strategies (e.g., "best of N", majority voting) have limitations in discerning the optimal solution, as shown in Figure 1. Empirical evidence suggests these methods often prioritize model confidence over the actual quality of the fix. Therefore, the crucial challenge is accurately identifying the most correct repair from TTS-generated candidates.

To address challenges in fault localization for real-world software development, we propose leveraging a specifically trained reward model to evaluate multiple fault localization candidates from test time scaling. A scoring mechanism based on a reward model can provide a more reliable basis for selection by quantifying each candidate's quality. Our designed reward model, **Hi**erarchical **Lo**calization **R**eward **M**odel (HiLoRM), assesses fault localization accuracy, enabling detailed comparative analysis at file, function, and line levels to identify the most precise answer from multiple candidates. This selection mechanism allows more reliable determination of the optimal fault localization result.

For training our reward model, we constructed HiFL-44k, a novel dataset with fine-grained bug localization data (file, function, line levels). We performed supervised fine-tuning (SFT) on a base model, then trained our reward model using reinforcement learning, specifically the Group Relative Policy Optimization (GRPO) method. During training, the model directly outputted the index of the most likely correct answer from multiple candidates. Experimental results show HiLoRM, trained on HiFL-44k, significantly improves file-level bug localization performance on SWE-Bench-Lite by 9.33% compared to scenarios without a reward model.

Our primary contributions are:

- A dedicated reward model for evaluating fault localization prediction effectiveness for GitHub issues.
- The open-source HiFL-44k dataset (44k instances of fine-grained bug localization data) for training repository-level bug localization models.
- Experimental results showing our trained reward model outperforms comparison models on repository-level fault localization.
- HiLoRM can be adapted to other fault location methods and is transferable.

2 Related Work

2.1 Automated Program Repair for Repository-Level Issues

The introduction of the SWE-Bench benchmark dataset (Jimenez et al., 2024) has spurred increased focus on Automated Program Repair (APR) techniques that more accurately reflect real-world software development scenarios. To address the complexities of repository-level repairs, two primary categories of approaches have emerged: pipeline-based and agent-based methods.

Agent-based methods assign distinct roles to LLMs, empowering them to autonomously perform tasks such as code analysis and repair. Prominent examples include SWE-agent (Yang et al., 2024c), CodeR (Chen et al., 2024), and MAGIS (Tao et al., 2024). This paradigm seeks to harness the autonomy and creative potential of LLMs for tackling intricate repair challenges. Conversely, pipeline-based methods employ predefined, structured repair workflows. These approaches, such as Agentless (Xia et al., 2024) and AutoCodeRover (Zhang et al., 2024), guide LLM behavior through explicit steps, thereby enhancing the controllability and predictability of the repair process. Although such methods constrain LLM autonomy, they can offer more directed guidance and may achieve greater efficiency in specific scenarios.

2.2 Fault Localization

Efficient and accurate fault localization is a pivotal aspect of software development and a crucial precursor to effective program repair. While traditional techniques like Spectrum-based Fault Localization (SBFL) and Mutation-based Fault Localization (MBFL) have proven effective, they often demand extensive high-quality labeled data and entail complex program analysis or test case generation processes. The demonstrated proficiency of LLMs in code understanding and reasoning has recently opened novel avenues for automated fault localization.

Initial research in LLM-based fault localization predominantly focused on smaller code segments, utilizing datasets such as Refactory and HumanEval for evaluation (Hu et al., 2019; Widyasari et al., 2024; Yang et al., 2024a). As the field has matured, repository-level fault localization has garnered significant attention due to its enhanced relevance to practical software development. Current strategies in this domain leverage LLMs through various means, including multi-stage localization frameworks (Qin et al., 2025), agent-based systems (Qin et al., 2024; Xu et al., 2025), meticulous fine-tuning or sophisticated prompt engineering (Chang et al., 2025), knowledge enhancement techniques, and graph-based methodologies (Yue et al., 2025). These approaches commonly embody "divide and conquer" principles and "fine-grained enhancement" tactics. A primary motivation for these strategies is to navigate the inherent context window limitations of LLMs by decomposing complex repository-wide localization tasks into more manageable sub-problems.

Our research builds upon these existing strengths, proposing a hierarchical, fine-grained repository-level fault localization framework. This framework leverages a coarse-to-fine strategy and test-time scaling to improve performance without requiring model fine-tuning.

3 Method

3.1 Motivation

The agentless paradigm offers a promising direction for simplifying software engineering automation, avoiding the complexities and overhead associated with full-fledged agent-based systems by directly leveraging LLMs. However, to effectively apply LLMs to pinpoint defects accurately and efficiently within vast code repositories without resorting to complex agentic control, a structured and sophisticated localization strategy is crucial; simpler agentless techniques might struggle to systematically narrow the search space or make optimal use of diverse LLM outputs.

To address this, our method introduces a significantly enhanced multi-stage hierarchical localization process that progresses through different levels: File, Function, and Line-level, as shown in Figure 2, named as **Hi**erarchical **F**ault **L**ocalization method (HiFL). Our core refinement to agentless localization lies in a systematic "sample-and-select" strategy employed at each hierarchical stage. Instead of relying on single LLM outputs or basic heuristics potentially used in more rudimentary agentless setups, we first leverage the LLM to generate a diverse set of candidate localizations (such as combinations of relevant files, specific functional units, or precise code lines) through multiple sampling. Subsequently, a dedicated reward model, HiLoRM, rigorously evaluates these varied proposals to identify and select the most promising candidate. This iterative application of LLM-driven diverse sampling coupled with HiLoRM-guided selection is designed to significantly enhance localization precision and reliability, allowing for a more robust exploration and exploitation of the LLM's capabilities within a streamlined agentless framework.

3.2 File-level Localization

The goal of the file localization stage is to identify a subset of files from the repository that are most relevant to the reported issue. This process consists of three key steps:

Relevant File Prediction First, we leverage the powerful code understanding and generation capabilities of LLMs. Given an issue description $\mathcal{I}$ and the tree structure of the repository $\mathcal{T}$, the LLM generates k candidate combinations of relevant files $\mathbb{C}_{rel} = \{C_1, C_2, \ldots, C_k\}$ through multiple sampling, where each $C_j \subset \mathcal{F}$ ($\mathcal{F}$ being the complete set of files in the repository).

Subsequently, HiLoRM will evaluates the entire set of candidate combinations $\mathbb{C}_{rel}$. This model receives the task context (i.e., $\mathcal{I}$ and $\mathcal{T}$) as well as all candidate combinations $\mathbb{C}_{rel}$, and from them selects the one combination that best represents the relevant files, denoted as F_{rel}:

$$F_{rel} = \text{Select}_{\text{HiLoRM}}(\mathcal{I}, \mathcal{T}, \mathbb{C}_{rel}) \quad (1)$$

where $\text{Select}_{\text{HiLoRM}}$ represents the selection function of the reward model HiLoRM.

Irrelevant File Prediction To further narrow down the search space, a similar strategy is employed to predict files that are irrelevant to the issue. The LLM, based on the issue description $\mathcal{I}$ and the codebase structure $\mathcal{T}$, generates a set of m candidate irrelevant file combinations $\mathbb{U}_{irrel} = \{U_1, U_2, \ldots, U_m\}$. The reward model HiLoRM also receives the task context and all candidate combinations $\mathbb{U}_{irrel}$, and from them selects one combination that best represents the set of irrelevant files F_{irrel}:

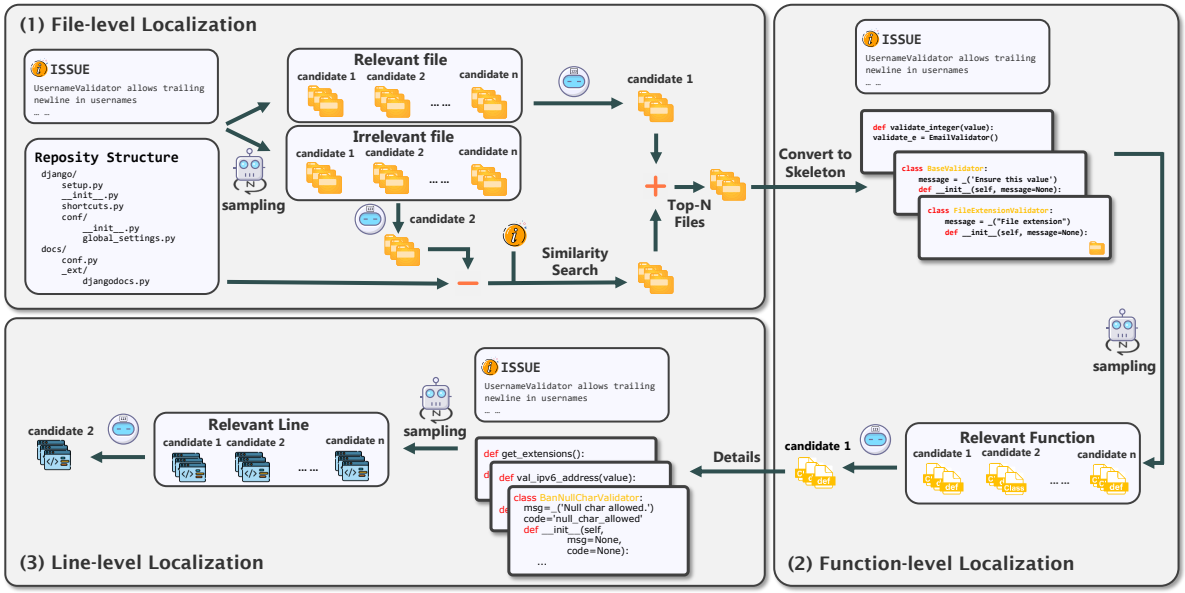


Figure 2: Overview of the proposed method, HiFL, which comprises three stages: file-level localization, function-level localization, and line-level localization.

$$F_{irrel} = \text{Select}_{\text{HiLoRM}}(\mathcal{I}, \mathcal{T}, \mathbb{U}_{irrel}) \quad (2)$$

These files will be excluded from the candidate set in the subsequent similarity retrieval process.

Similarity Retrieval After excluding the predicted irrelevant files, we obtain a reduced candidate file set $\mathcal{F}' = \mathcal{F} \setminus F_{irrel}$. In this stage, we calculate the semantic similarity between the issue description $\mathcal{I}$ and each file f in $\mathcal{F}'$. Specifically, we use the `text-embedding-3-small`[1] model to convert the issue description $\mathcal{I}$ and the full content of each file f into high-dimensional vector embedding representations. Let $\text{emb}(\cdot)$ be the embedding function provided by `text-embedding-3-small`, then the embedding of the issue description is $v_{\mathcal{I}} = \text{emb}(\mathcal{I})$, and the embedding of each file f is $v_f = \text{emb}(f)$.

Subsequently, we evaluate their relevance $S_{cos}(\mathcal{I}, f)$ by calculating the cosine similarity between these vectors:

$$S_{cos}(\mathcal{I}, f) = \frac{v_I \cdot v_f}{\|v_{\mathcal{I}}\| \|v_f\|} \quad (3)$$

where $v_{\mathcal{I}} \cdot v_f$ is the dot product of the vectors, and $\|v_{\mathcal{I}}\|$ and $\|v_f\|$ are the Euclidean norms (L2 norms) of vectors $v_{\mathcal{I}}$ and v_f respectively.

The top N files with the highest cosine similarity to the issue description, denoted as F_{sim}, are selected. Finally, the output of the file localization stage, F_{final}, is the union of the set of top_N predicted relevant files F_{rel} and the set of top_N files from the similarity retrieval F_{sim}:

$$F_{final} = \text{top\_N}(F_{rel}) \cup \text{top\_N}(F_{sim}) \quad (4)$$

3.3 Function-level Localization

After determining the target file set F_{final}, we further localize the specific code units (such as functions, classes, methods, or variables) within these files that may contain the erroneous code.

Code Skeletonization First, we convert each file in $\mathcal{F}_{final}$ into its "skeleton representation" f_{skel}. This representation preserves the structural information of the code while omitting implementation details.

Candidate Code Unit Generation and Selection Given the issue description $\mathcal{I}$ and the skeleton representation of a file, f_{skel}, the LLM generates a set of p candidate code unit sets, $\mathbb{E}_{cand} = \{E_1, E_2, \ldots, E_p\}$, where each E_q contains several code unit signatures from within f_{skel}.

The reward model HiLoRM receives the task context ($\mathcal{I}$, f_{skel}) and all candidate sets $\mathbb{E}_{cand}$, and from these, it selects the set of code units E_{target} most likely requiring modification, where:

$$E_{target} = \text{Select}_{\text{HiLoRM}}(\mathcal{I}, f_{skel}, \mathbb{E}_{cand}) \quad (5)$$

This process is carried out for each file in F_{final}.

[1] https://openai.com/api/

3.4 Line-level Localization

Line-level localization is the final stage of fault localization, aiming to precisely pinpoint the specific lines of code that need modification.

Based on the target code units E_{target} localized in the previous stage, we extract their complete contextual information, Context(E_{target}), including the line number.

We provide the issue description $\mathcal{I}$ and Context(E_{target}) to the LLM. The LLM, through multiple sampling, generates a collection of r candidate sets of code lines (or line number ranges), $\mathbb{L}_{cand} = \{L_1, L_2, \ldots, L_r\}$.

HiLoRM will receives the task context ($\mathcal{I}$, Context(E_{target})) and all candidate code line sets $\mathbb{L}_{cand}$, and from these, selects a final set of code lines most likely requiring modification, L_{final}:

$$L_{final} = \text{Select}_{\text{HiLoRM}}(\mathcal{I}, \text{Context}(E_{target}), \mathbb{L}_{cand}) \tag{6}$$

Through this three-stage, coarse-to-fine localization strategy, combined with the generative capabilities of LLMs and the ability of reward models to select the optimal choice from multiple candidates, our method aims to efficiently and accurately pinpoint code defects.

3.5 Reward Function

The reward function of the GRPO training phase consists of two components, each assigned a weight of 1.0: Correctness of Selection (CS) and a format reward. The format reward yielded 1.0 for outputs adhering to the predefined format and 0 otherwise. The CS component was calculated as:

$$S_{CS} = \begin{cases} 1, & \text{if } pred_{index} = truth_{index} \\ 0, & \text{if } pred_{index} \neq truth_{index} \end{cases} \tag{7}$$

where $pred_{index}$ is the model's predicted index and $truth_{index}$ is the ground truth index.

4 HiFL-44k Dataset

This study constructed a large-scale fault localization dataset comprising 44k instances, named HiFL-44k, the statistical results of the data set are shown in Table 1. These instances were sourced from nearly 100 open-source GitHub repositories characterized by high star ratings (GitHub Stars > 1,000) and recent activity (updated within the last two months). To ensure the reliability and quality of the dataset, stringent filtering criteria were established: relevant issues must have been resolved and closed via a single merged Pull Request (PR); concurrently, to ensure analytical validity, repositories containing only a small number of Python source files (`.py`) were excluded. To prevent the data leakage, any GitHub repositories included in SWE-Bench or its sub-benchmarks were excluded during the construction of HiFL-44k. Subsequently, through meticulous parsing of the patch information contained within these PRs, we established file-level, function-level, and line-level ground-truth modifications.

Table 1: Detailed statistics of the HiFL-44k dataset

Category	File-level	Function-level	Line-level
Number	27,425	9,193	7,484

To evaluate model performance under conditions more aligned with real-world application scenarios, we employed the Qwen2.5-Coder-7B-Instruct model to predict relevant and irrelevant files. After deduplicating the prediction results, Precision (P), Recall (R), and a weighted F1 score ($F1_w$) were adopted as core evaluation metrics. Inspired by the work of (Ma et al., 2025), and prioritizing the model's performance in fault recall to mitigate the risk of overlooking actual fault locations, this study adopted the following weighted F1 calculation formula:

$$F1_w = \lambda \cdot R + (1 - \lambda) \cdot F_{\text{custom}} \tag{8}$$

$$F_{\text{custom}} = (1 + \beta^2) \frac{P \cdot R}{(\beta^2 \cdot P) + R} \tag{9}$$

where the λ, β are hyperparameters used to balance the impact of the recall and precision on the final score, we set $\lambda = 0.8$ and $\beta = 3$ in this paper. This formula emphasizes recall performance during evaluation by assigning a higher weight to Recall.

In the experimental design for function-level and line-level fault localization, we first conducted a full parse of the target repository to construct a suspect pool containing all potentially modifiable code units (e.g., functions, classes, methods, variable declarations, and executable code lines). Thereafter, we mixed the ground-truth fault locations with other code elements from this suspect pool at various predefined ratios. This process generated prediction candidate sets with different diffi-

culty gradients, intended to comprehensively assess the model's localization capabilities across diverse complex scenarios.

5 Experiment

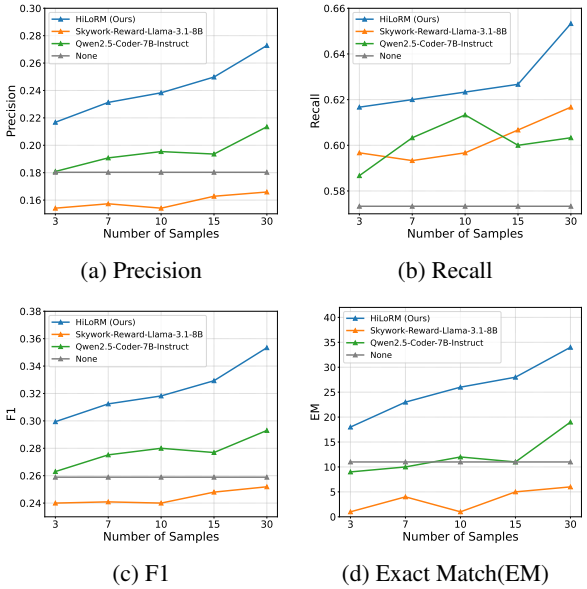
5.1 Model, Data, and Training Phases

The `Qwen2.5-Coder-7B-Instruct` model (Hui et al., 2024; Yang et al., 2024b) served as the backbone network for HiLoRM. Training data was sourced from a subset of points extracted from each of four different problem types within the constructed HiFL-44k dataset; these points were subsequently combined and shuffled. The entire training process was managed using the ms-Swift(Zhao et al., 2025). An initial SFT phase was conducted to adapt the base model to the required output format, utilizing a dataset of 1,000 data points from HiFL-44k. Following SFT, the GRPO training stage was performed using DeepSpeed ZeRO3 and LoRA, with the LoRA rank set to 16 and LoRA alpha to 32. Training was conducted for 1 epoch using a cosine learning rate decay strategy, with an initial learning rate of 5e-6 and a warmup ratio of 0.01. The batch size was 2, and the group size was 4. To optimize memory, bf16 mixed-precision training was enabled. The training procedure required approximately 12 hours on four RTX4090 GPUs.

5.2 Evaluation Methodology

This section outlines the setup for assessing the performance of the trained HiLoRM. Please refer to Appendix A for the detailed calculation methods of the evaluation metrics, the experimental results are the average of five experiments.

Benchmarking Setup The primary evaluation was conducted on the SWE-Bench-Lite dataset[2], which is a subset of SWE-Bench[3] containing 300 instances (Jimenez et al., 2024). For performance comparison, HiLoRM was benchmarked against `Skywork-Reward-Llama-3.1-8B`[4] (Liu and Zeng, 2024). An ablation study was also performed by comparing HiLoRM against the original Qwen2.5-Coder-7B-Instruct model. During these evaluations, the number of sampling for the LLM generating prediction results was varied across five values: 3, 7, 10, 15, and 30. A higher temperature of 1.0 was used for the LLM to ensure a maximally diverse set of candidate solutions.



(a) Precision (b) Recall

(c) F1 (d) Exact Match(EM)

Figure 3: Performance of Qwen2.5-Coder-7B-Instruct as an inference model in the relevant file localization task (Gray line: results without any reward model and no multiple sampling).

Generalizability Assessment To further verify the transferability and efficacy of HiLoRM, supplementary experiments were conducted on the CoSIL method (Jiang et al., 2025).

CoSIL is a training-free, LLM-driven fault localization method that dynamically constructs code graphs and utilizes module call graph enhanced reflection, iterative search, and a pruner to precisely constrain the search space and manage context, without requiring pre-built indexes.

6 Main Result

6.1 Effectiveness

In order to evaluate the effectiveness of a novel proposed method for code defect localization tasks in software engineering. Experiments were conducted on the SWE-Bench-Lite dataset using the Qwen2.5-Coder-7B-Instruct and Qwen2.5-Coder-32B-Instruct large language models. The reward models selected were the existing Skywork-Reward-Llama-3.1-8B and HiLoRM, a model trained in this work. The core evaluation methodology employed was HiFL. The recall results are shown in Table 2, and the precision results are shown in Table A1 in the Appendix B.2.

Experimental results on the SWE-Bench-Lite dataset show that when using Qwen2.5-Coder-7B-

[2]https://huggingface.co/datasets/princeton-nlp/SWE-bench_Lite

[3]https://huggingface.co/datasets/SWE-bench/SWE-bench

[4]https://www.modelscope.cn/models/Skywork/Skywork-Reward-Models

Table 2: Comparison of fault localization recall performance under different reward models and numbers of samples (3, 7, 10, 15, and 30) in the HiFL method (None: No reward model used, number of samples = 1).

Reward Model	Qwen2.5-Coder-7B-Instruct					Qwen2.5-Coder-32B-Instruct				
	3	**7**	**10**	**15**	**30**	**3**	**7**	**10**	**15**	**30**
File-level localization										
None	0.79	0.79	0.79	0.79	0.79	0.86	0.86	0.86	0.86	0.86
Skywork-Reward -Llama-3.1-8B	0.7967	0.8	0.7967	0.8067	0.8167	0.8567	0.8533	0.8633	0.8667	0.8667
HiLoRM(Ours)	**0.8033**	**0.8067**	**0.8133**	**0.8133**	**0.82**	**0.87**	**0.8667**	**0.87**	**0.8733**	**0.8767**
Function-level localization										
None	**0.2486**	0.2486	0.2486	0.2486	0.2486	**0.3410**	0.3410	0.3410	0.3410	0.3410
Skywork-Reward -Llama-3.1-8B	0.2414	0.2537	0.2566	0.2375	0.2371	0.2915	0.2492	0.2507	0.2648	0.2390
HiLoRM(Ours)	0.2485	**0.2735**	**0.2743**	**0.2682**	**0.2771**	0.3311	**0.3446**	**0.3599**	**0.3731**	**0.3873**
Line-level localization										
None	0.12	0.12	0.12	**0.12**	0.12	0.1939	0.1939	0.1939	0.1939	0.1939
Skywork-Reward -Llama-3.1-8B	0.1194	0.1183	0.0944	0.1033	0.0567	0.19	0.1928	**0.2217**	0.2122	0.1767
HiLoRM(Ours)	**0.1206**	**0.1294**	**0.1344**	0.1078	**0.1228**	**0.1992**	**0.2086**	0.2064	**0.2361**	**0.2269**

Instruct as the inference model, the HiFL method, which uses HiLoRM as the reward model, improves code fault localization recall by 3% at the file level, 11% at the function level, and 10% at the line level, compared to a baseline configuration without a reward model and using multi-round sampling. This indicates that the proposed method can effectively enhance the model's localization capabilities through multi-round sampling and a reward-model-guided candidate answer selection mechanism.

Delving deeper into the file-level localization sub-task, our method achieved optimal or near-optimal performance across all metrics, including Precision, Recall, F1-score, and Exact Match (Figure 3 and Appendix B.1). A clear positive correlation was observed between the number of sampling iterations and performance. Notably, on the Exact Match metric, HiLoRM identified up to 34 correct instances, a stark contrast to the 10 instances under the no-sampling condition and the single-digit results from the Skywork-Reward-Llama-3.1-8B model. Furthermore, our method yielded a significant precision uplift, ranging from 20.2% to 51% over the baseline.

A key finding is that although the HiLoRM reward model was trained based on a 7B parameter-scale model, it can still effectively evaluate and filter prediction results generated by larger base models (such as a 32B model), demonstrating good generalization capabilities.

We conducted an independent t-test comparing HiLoRM against the two baseline models (Skywork-Reward-Llama-3.1-8B and Qwen2.5-Coder-7B-Instruct), as shown in Appendix B.4.

To validate the superiority of our task-specific training approach, we benchmarked HiLoRM against RISE-Judge-Qwen2.5-7B[5] (Yu et al., 2025), a reward model trained using the DPO technique. As shown in Table A2, HiLoRM consistently outperforms RISE-Judge on the file-level localization task, confirming the advantages of our specialized training methodology.

Finally, to assess the method's robustness on more challenging, real-world problems, we conducted experiments on the SWE-Bench-Verified [6] (OpenAI, 2024) dataset (see Appendix C). The results corroborated our findings, with HiLoRM again achieving the highest recall at the file (6 – 7%) and function (5%) levels compared to the baseline. This reinforces the efficacy of our approach in complex, practical scenarios.

6.2 Transferability

To demonstrate the transferability of our proposed method, we evaluated it from two perspectives: its

[5]https://huggingface.co/R-I-S-E/RISE-Judge-Qwen2.5-7B

[6]https://huggingface.co/datasets/princeton-nlp/SWE-bench_Verified

Table 3: Recall performance comparison of the CoSIL method: No reward model (None) with number of samples = 1 vs. HiLoRM with various numbers of samples

Reward Model	#sample	Qwen2.5-Coder-7B-Instruct		
		TOP-1	TOP-3	TOP-5
File-level localization				
None	1	0.4633	0.5767	0.59
HiLoRM(Ours)	3	0.4767	0.6033	0.6133
	7	0.4967	0.61	0.6233
	10	0.4867	0.6167	0.6367
Function-level localization				
None	1	0.1867	0.2433	0.2567
HiLoRM(Ours)	3	0.1933	0.2567	0.2667
	7	0.1967	0.2533	0.2667
	10	0.2133	0.2733	0.29

Table 4: Relevant-file-level localization recall of HiFL on SWE-Bench-Lite using YiRM and its base model.

Reward Model	Qwen2.5-Coder-7B-Instruct				
	3	7	10	15	30
Yi-Coder-9B	0.5033	0.49	0.4967	0.4867	0.4933
YiRM	**0.58**	**0.5967**	**0.5933**	**0.5833**	**0.6033**

Table 5: File-Level localization recall of HiFL on SWE-Bench-Lite using YiRM and its base model.

Reward Model	Qwen2.5-Coder-7B-Instruct				
	3	7	10	15	30
Yi-Coder-9B	0.77	0.7767	0.7833	0.79	0.77
YiRM	**0.7967**	**0.79**	**0.8**	**0.7933**	**0.7867**

applicability to different localization frameworks and its generalizability across diverse foundational model architectures.

First, to validate the transferability of the HiLoRM reward model trained in this study, its performance was further evaluated within the CoSIL method framework, with experimental results detailed in Table 3. The experiments compared the performance of the CoSIL method under two configurations: one without a reward model, and another using HiLoRM to select what it deems the most promising candidate answer. Three different sampling numbers were investigated: 3, 7, and 10. Given that the CoSIL method primarily focuses on file-level and function-level localization, this study also concentrated on these two levels for evaluation, adopting the TOP-N accuracy metric as used in the original CoSIL publication.

The experimental results indicate that when Qwen2.5-Coder-7B-Instruct served as the base inference model, the CoSIL method integrated with HiLoRM achieved a performance improvement of up to 7.9% in TOP-N accuracy for file-level localization, and up to 13.0% for function-level localization, compared to the CoSIL method without a reward model. These findings demonstrate that although the training data for HiLoRM was primarily generated based on the design principles and processes of an agentless method, the reward model exhibits good transferability, effectively empowering and enhancing the performance of other distinct code defect localization approaches.

In addition to its transferability across methods, we investigated the generalizability of our approach across different LLM architectures. To this end, we applied our fine-tuning strategy to a LLaMA-based model, Yi-Coder-9B[7](Young et al., 2024), creating a new reward model variant (YiRM), the results are shown in Table 4 and Table 5. The results reveal a notable 20% absolute improvement in localization when the search space is confined to relevant files, a substantial gain that underscores the effectiveness of our method. In contrast, the performance gains are more modest when the search includes irrelevant files, due to the added complexity of the similarity-based retrieval stage.

Experiments result on Qwen- and LLaMA-based models show that our method generalizes effectively across diverse LLM architectures. This validates the broader applicability of our approach beyond any single model family.

In conclusion, the ability of our reward model to enhance a distinct localization framework (CoSIL) and its successful implementation on a different foundational architecture (LLaMA-based) collectively validate the applicability and transferability of our proposed method.

6.3 Ablation Study

Finally, to verify that the performance gains of the HiLoRM model trained in this study are not solely attributable to the inherent capabilities of its base model, we further compared the performance differences between HiLoRM and its base model within the HiFL method framework. Detailed experimental results are presented in Table 6.

The experimental results indicate that although the base model of HiLoRM already outperforms the comparative reward model Skywork-Reward-Llama-3.1-8B on most evaluation met-

[7]https://huggingface.co/01-ai/Yi-Coder-9B

Table 6: Comparison of fault localization recall performance under different reward models and numbers of samples (3, 7, 10, 15, and 30) in the HiFL method (using HiLoRM and its base model).

Reward Model	Qwen2.5-Coder-7B-Instruct					Qwen2.5-Coder-32B-Instruct				
	3	**7**	**10**	**15**	**30**	**3**	**7**	**10**	**15**	**30**
File-level localization										
Qwen2.5-Coder-7B-Instruct	0.7967	0.7967	0.8033	0.7933	0.79	0.8567	0.85	0.86	0.8533	0.8533
HiLoRM(Ours)	**0.8033**	**0.8067**	**0.8133**	**0.8133**	**0.82**	**0.87**	**0.8667**	**0.87**	**0.8733**	**0.8767**
Function-level localization										
Qwen2.5-Coder-7B-Instruct	**0.2524**	0.2546	0.2315	0.2327	0.2336	**0.3422**	0.3163	0.3356	0.3448	0.3020
HiLoRM(Ours)	0.2485	**0.2735**	**0.2743**	**0.2682**	**0.2771**	0.3311	**0.3446**	**0.3599**	**0.3731**	**0.3873**
Line-level localization										
Qwen2.5-Coder-7B-Instruct	**0.1361**	0.1144	0.1222	**0.1328**	0.0867	**0.2103**	**0.2094**	0.1894	0.1825	0.1856
HiLoRM(Ours)	0.1206	**0.1294**	**0.1344**	0.1078	**0.1228**	0.1992	0.2086	**0.2064**	**0.2361**	**0.2269**

rics, HiLoRM itself consistently surpasses its base model (i.e., a configuration relying only on the base model with multi-round sampling and selection of the best result, without HiLoRM) across all assessed aspects. When Qwen2.5-Coder-7B-Instruct was used as the base model, its specific performance in the relevant file localization task is also illustrated in Figure 3. This figure shows that HiLoRM outperforms the direct application of its base model on all four evaluation metrics, notably achieving improvements of 19.8% to 29% in precision and F1-score, and a significant gain of over 100% in the exact match metric.

Furthermore, our evaluation on SWE-Bench-Verified corroborates these positive findings, demonstrating that the performance of HiLoRM remains robust (see Appendix C).

Therefore, these results robustly demonstrate that the superior performance exhibited by the HiLoRM model is not merely a consequence of the underlying strength of the base model employed, but rather that HiLoRM contributes substantial additional improvements to the localization task through its learned reward mechanism.

7 Conclusion

To address the context limitations and candidate selection challenges faced by LLMs in software repository-level fault localization, this paper proposes a hierarchical localization reward model, HiLoRM. It was trained on the HiFL-44k dataset that we built, can evaluate and select LLM-generated candidates at the file, function, and line levels. Experiments demonstrate that integrating HiLoRM significantly improves the fault localization accuracy of LLMs on SWE-Bench-Lite (especially at the file level) and effectively selects the best candidates from multiple samples. HiLoRM also exhibits good generalization to larger-scale base models and different methods such as CoSIL. Its performance improvement is primarily attributed to the reward mechanism, rather than just the base model.

HiLoRM and its accompanying HiFL-44k dataset offer an effective solution for precise fault localization by LLMs by optimizing candidate selection. This significantly enhances localization accuracy and reliability, and holds important practical significance.

Limitations

Despite promising results, this study has limitations warranting future work: First, computational constraints limited exploration of HiLoRM built upon larger foundation models; their performance characteristics remain unassessed. Second, incorporating deeper reasoning or multi-step verification into HiLoRM's selection process, potentially via chain-of-thought or iterative refinement, could improve performance beyond direct index output. Finally, the HiFL-44k dataset, while substantial, may not cover all real-world fault complexities. Expanding its scope or testing HiLoRM on more intricate bug scenarios is a key future direction.

Acknowledgments

This work is supported by thet National Natural Science Foundation of China (Grant No. 62276171, 62476173, 62532007), Guangdong Basic and Applied Basic Research Foundation (Grant No. 2024A1515011938 and 2020B1515120028),

Shenzhen Fundamental Research-General Project (Grant No. JCYJ20240813141503005 and JCYJ20240813142610014), Major Special Project for Philosophy and Social Sciences Research of the Ministry of Education (Grant No. 2025JZDZ010), CCF-Huawei Populus Grove Fund(CCF-HuaweiFM2024004), The Graduate Students' Project of SZU (868-000002020234).

References

Jianming Chang, Xin Zhou, Lulu Wang, David Lo, and Bixin Li. 2025. Bridging bug localization and issue fixing: A hierarchical localization framework leveraging large language models. *arXiv preprint arXiv:2502.15292*.

Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, and 1 others. 2024. Coder: Issue resolving with multi-agent and task graphs. *arXiv preprint arXiv:2406.01304*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Yang Hu, Umair Z. Ahmed, Sergey Mechtaev, Ben Leong, and Abhik Roychoudhury. 2019. Refactoring based program repair applied to programming assignments. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 388–398.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Zhonghao Jiang, Xiaoxue Ren, Meng Yan, Wei Jiang, Yong Li, and Zhongxin Liu. 2025. Cosil: Software issue localization via llm-driven code repository graph searching. *Preprint*, arXiv:2503.22424.

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

Bingchang Liu, Chaoyu Chen, Zi Gong, Cong Liao, Huan Wang, Zhichao Lei, Ming Liang, Dajun Chen, Min Shen, Hailian Zhou, and 1 others. 2024. Mftcoder: Boosting code llms with multitask fine-tuning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5430–5441.

Chris Yuhao Liu and Liang Zeng. 2024. Skywork reward model series. `https://huggingface.co/Skywork`.

Zexiong Ma, Chao Peng, Pengfei Gao, Xiangxin Meng, Yanzhen Zou, and Bing Xie. 2025. Sorft: Issue resolving with subtask-oriented reinforced fine-tuning. *Preprint*, arXiv:2502.20127.

OpenAI. 2024. Introducing swe-bench verified. Accessed: 2025-06-27.

Yihao Qin, Shangwen Wang, Yiling Lou, Jinhao Dong, Kaixin Wang, Xiaoling Li, and Xiaoguang Mao. 2024. Agentfl: Scaling llm-based fault localization to project-level context. *arXiv preprint arXiv:2403.16362*.

Yihao Qin, Shangwen Wang, Yiling Lou, Jinhao Dong, Kaixin Wang, Xiaoling Li, and Xiaoguang Mao. 2025. SoapFL: A Standard Operating Procedure for LLM-Based Method-Level Fault Localization . *IEEE Transactions on Software Engineering*, 51(04):1173–1187.

Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. 2024. Magis: Llm-based multi-agent framework for github issue resolution. In *Advances in Neural Information Processing Systems*, volume 37, pages 51963–51993. Curran Associates, Inc.

Nalin Wadhwa, Atharv Sonwane, Daman Arora, Abhav Mehrotra, Saiteja Utpala, Ramakrishna B Bairi, Aditya Kanade, and Nagarajan Natarajan. 2024. Masai: Modular architecture for software-engineering ai agents. In *NeurIPS 2024 Workshop on Open-World Agents*.

Zhijie Wang, Zijie Zhou, Da Song, Yuheng Huang, Shengmai Chen, Lei Ma, and Tianyi Zhang. 2025. Towards understanding the characteristics of code generation errors made by large language models. *Preprint*, arXiv:2406.08731.

Ratnadira Widyasari, Jia Wei Ang, Truong Giang Nguyen, Neil Sharma, and David Lo. 2024. Demystifying faulty code with llm: Step-by-step reasoning for explainable fault localization. *Preprint*, arXiv:2403.10507.

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

Chuyang Xu, Zhongxin Liu, Xiaoxue Ren, Gehao Zhang, Ming Liang, and David Lo. 2025. Flexfl: Flexible and effective fault localization with open-source large language models. *IEEE Transactions on Software Engineering*, pages 1–17.

Aidan Z. H. Yang, Claire Le Goues, Ruben Martins, and Vincent Hellendoorn. 2024a. Large language models for test-free fault localization. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA. Association for Computing Machinery.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024b. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

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

Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, Kaidong Yu, Peng Liu, Qiang Liu, Shawn Yue, Senbin Yang, Shiming Yang, Tao Yu, Wen Xie, Wenhao Huang, and 11 others. 2024. Yi: Open foundation models by 01.ai. *CoRR*, abs/2403.04652.

Jiachen Yu, Shaoning Sun, Xiaohui Hu, Jiaxu Yan, Kaidong Yu, and Xuelong Li. 2025. Improve llm-as-a-judge ability as a general ability. *Preprint*, arXiv:2502.11689.

Li Yue, Liu Bohan, Zhang Ting, Wang Zhiqi, Lo David, Yang Lanxin, Lyu Jun, and Zhang He. 2025. A knowledge enhanced large language model for bug localization. In *2025 ACM International Conference on the Foundations of Software Engineering (FSE)*, FSE '25, Trondheim, Norway. Association for Computing Machinery.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, page 1592–1604, New York, NY, USA. Association for Computing Machinery.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. 2025. Swift: A scalable lightweight infrastructure for fine-tuning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(28):29733–29735.

A Evaluation index calculation method

A.1 File-level

In file-level localization tasks, we employ Precision, Recall, and F1-score as core evaluation metrics to measure the model's performance in identifying relevant files. Precision quantifies the proportion of files predicted as relevant that are actually relevant. Recall is defined as the proportion of all genuinely relevant files that are successfully identified and retrieved by the model. The F1-score, as the harmonic mean of Precision and Recall, provides a consolidated reflection of the model's overall performance.

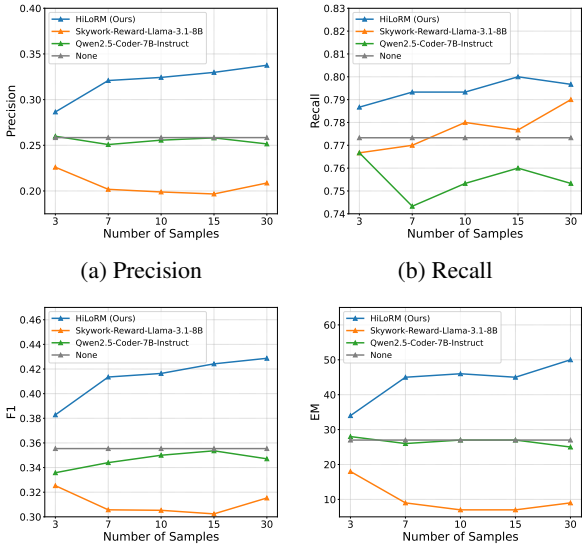
Furthermore, we incorporate the Exact Match, which refers to the proportion of instances where the set of files predicted by the model perfectly matches the actual set of relevant files. Collectively, these metrics offer a comprehensive assessment of the model's performance in the file localization task from the perspectives of both accuracy and completeness. For the task of localizing relevant files within this paper, these are precisely the metrics utilized to evaluate its effectiveness.

A.2 Function-level

For function-level localization, we continue to employ the three core metrics introduced in the previous section: Precision, Recall, and F1-score for evaluation. Acknowledging the inherent uncertainty in the outputs from LLMs—for instance, while the ground truth for function-level localization of a specific instance might be `valid.metric` (class_name.function_name), an LLM might sometimes output only the class name `valid` without further specifying the concrete function. Within our evaluation criteria in this paper, we uniformly consider both scenarios (i.e., returning the full function signature or only the class name) as valid and accurate localizations.

A.3 Line-level

we employ a hierarchical matching approach to calculate relevant performance metrics. The evaluation process first compares instance identifiers to ensure that predictions and ground truth refer to the same problem instance, then uses a normalization function to match file paths, ensuring correct identification of different path representations pointing to the same file. At the finest granularity of line level, the system compares the type of code elements (such as functions, classes, variables), their values (such as function names, class names), and positional information (line number ranges). When a predicted element matches a ground truth element in type and value, and their *line number ranges overlap*, it is classified as a true positive (TP). Based on the matching results, we calculate standard evaluation metrics:



(a) Precision (b) Recall

(c) F1 (d) Exact Match(EM)

Figure A1: Performance of Qwen2.5-Coder-32B-Instruct as an inference model in the relevant file localization task (Gray line: results without any reward model and no multiple sampling).

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

$$F1 = 2\frac{Precision \times Recall}{Precision + Recall} \tag{12}$$

where FP represents the number of incorrectly predicted elements, and FN represents the number of elements that should have been modified but were not predicted.

A simple example illustrates this: if the ground truth indicates that the `func_name` function in `file.py` (lines 1-10) should be modified, and the prediction is for the same function in the same file but with line range 2-10, this is classified as a true positive because the type and value match and the line ranges overlap. This evaluation method objectively reflects the actual performance of models in code element localization tasks.

Table A1: Comparison of fault localization precision performance under different reward models and numbers of samples (3, 7, 10, 15, and 30) in the HiFL method (None: No reward model used, number of samples = 1).

Reward Model	Qwen2.5-Coder-7B-Instruct					Qwen2.5-Coder-32B-Instruct				
	3	7	10	15	30	3	7	10	15	30
File-level localization										
None	0.1760	0.1760	0.1760	0.1760	0.1760	0.2584	0.2584	0.2584	0.2584	0.2584
Skywork-Reward -Llama-3.1-8B	0.1717	0.1734	0.1724	0.1737	0.1765	0.2260	0.2018	0.1989	0.1967	0.2087
Qwen2.5-Coder-7B-Instruct	0.1779	0.1769	0.1773	0.1764	0.1792	0.2599	0.2508	0.2556	0.2581	0.2515
HiLoRM(Ours)	**0.1835**	**0.1860**	**0.1877**	**0.1910**	**0.1948**	**0.2865**	**0.3209**	**0.3243**	**0.3298**	**0.3376**
Function-level localization										
None	0.1678	0.1678	0.1678	0.1678	0.1678	0.1743	0.1743	0.1743	0.1743	0.1743
Skywork-Reward -Llama-3.1-8B	0.1491	0.1287	0.1283	0.1158	0.1055	0.1307	0.0915	0.0943	0.0950	0.0718
Qwen2.5-Coder-7B-Instruct	**0.1857**	0.1859	0.1706	0.1485	0.1844	0.1827	0.1624	0.1838	0.1754	0.1374
HiLoRM(Ours)	0.1825	**0.2007**	**0.2157**	**0.1932**	**0.2062**	**0.197**	**0.2157**	**0.2159**	**0.1906**	**0.2094**
Line-level localization										
None	0.0989	0.0989	0.0989	0.0989	0.0989	0.1867	0.1867	**0.1867**	0.1867	0.1867
Skywork-Reward -Llama-3.1-8B	0.1	0.0711	0.07	0.0728	0.0467	0.1722	0.1517	0.1811	0.1744	0.1444
Qwen2.5-Coder-7B-Instruct	**0.1156**	0.0917	0.11	**0.1161**	0.0744	**0.1872**	0.1933	0.1839	0.1606	0.1597
HiLoRM(Ours)	0.1009	**0.1172**	**0.1178**	0.0961	**0.1058**	0.1744	**0.1973**	0.1861	**0.2150**	**0.2061**

B Other experimental results in SWE-Bench-Lite

B.1 Relevant File Localization Results

As depicted in Figure A1, when Qwen2.5-Coder-32B-Instruct is employed as the inference model, HiLoRM continues to demonstrate better performance across various evaluation metrics. Notably, its performance exhibits a trend of continuous improvement as the number of samples increases.

B.2 Precision Results

While Recall is often the prioritized metric in fault localization tasks, an excessively high Recall coexisting with overly low Precision can negatively impact the accuracy of the model's judgment, especially given the context length limitations of LLMs. Therefore, we have summarized and presented the Precision results for all experiments conducted on the SWE-Bench-Lite dataset in this paper in Table A1. As can be seen from the table, our method has relatively high Precision in most cases.

B.3 Comparison with an RLHF-Trained Reward Model

To further validate the novelty of the HiLoRM, we conducted a comparative analysis against RISE-Judge-Qwen2.5-7B. This model was selected as a representative baseline due to its training with Direct Preference Optimization (DPO), a prominent RLHF technique, and its strong performance on the RewardBench benchmark. As presented in Table A2, our model significantly outperforms RISE-Judge-Qwen2.5-7B on the SWE-Bench-Lite dataset for file-level localization, showing superior performance in both recall and precision. This result demonstrates that our task-tuned, hierarchical scorer provides a substantial advantage over a general-purpose generative judgment model for the specific challenge of fault localization.

Table A2: Recall and Precision Comparison on HiFL method with HiLoRM and RISE-Judge-Qwen2.5-7B

Reward Model	Qwen2.5-Coder-7B-Instruct				
	3	7	10	15	30
Recall for File-level localization					
RISE-Judge-Qwen2.5-7B	0.7967	0.79	0.8	0.7933	0.7867
HiLoRM (Ours)	**0.8033**	**0.8067**	**0.8133**	**0.8133**	**0.82**
Precision for File-level localization					
RISE-Judge-Qwen2.5-7B	0.1779	0.1790	0.1804	0.1777	0.1814
HiLoRM (Ours)	**0.1835**	**0.1860**	**0.1877**	**0.1910**	**0.1948**

B.4 Statistical Significance

To validate that the observed performance gains of HiLoRM are not a result of random chance, we conducted a statistical significance analysis. The analysis was performed on the results from five independent experimental replications. The infer-

Table A3: Statistical significance of HiFL with HiLoRM and two baseline models on SWE-Bench-Lite($p < 0.05$).

Reward model	3	7	10	15	30
Relevant-file-level localization					
Skywork-Reward-Llama-3.1-8B	0.0423	0.0301	0.0129	0.0241	0.0024
Qwen2.5-Coder-7B-Instruct	0.0018	0.0159	0.0148	0.0348	0.0004
Function-level localization					
Skywork-Reward-Llama-3.1-8B	0.6747	0.0227	0.0218	0.0101	0.0018
Qwen2.5-Coder-7B-Instruct	0.3123	0.0816	0.0239	0.0091	0.0002
Line-level localization					
Skywork-Reward-Llama-3.1-8B	0.1047	0.0487	0	0.5122	0
Qwen2.5-Coder-7B-Instruct	0.0037	0.0019	0.0206	0	0.0001

ence model for these experiments consistently used the Qwen2.5-Coder-7B-Instruct. We set the significance level ($P < 0.05$), a standard threshold for statistical validation.

The detailed results are presented in Table A3. The statistical analysis validates the effectiveness of HiLoRM. The method demonstrates statistically significant improvements over strong baselines in nearly all evaluated conditions, with particular strength in file-level localization and at larger sample sizes across all tasks.

C Results in SWE-Bench-Verified

As seen in A4, at the file and function levels, HiLoRM outperforms both baselines across all sampling settings, achieving the highest recall. Also, HiLoRM surpasses the no-reward baseline by 6–7% at the file level and by over 5% at the function level, indicating that our method can enhances localization performance in complex, real-world scenarios.

At the line-level, HiLoRM achieves the highest recall in lower sampling (3–10). The benefit decreases, in higher sampling (15,30), due to accumulated noise and increased difficulty in fine-grained ranking.

Overall, HiLoRM remains effective on more challenging datasets, supporting its applicability in practical fault localization scenarios.

Table A4: Recall performance on SWE-Bench-Verified in the HiFL method under different reward models and numbers of samples (3, 7, 10, 15, and 30) (None: No reward model used, number of samples = 1) .

Reward Model	Qwen2.5-Coder-7B-Instruct				
	3	7	10	15	30
File-level localization					
None	0.5387	0.5387	0.5387	0.5387	0.5387
Skywork-Reward-Llama-3.1-8B	0.5351	0.5430	0.5418	0.5386	0.5836
Qwen2.5-Coder-7B-Instruct	0.5501	0.5599	0.5644	0.5544	0.5699
HiLoRM(Ours)	**0.5707**	**0.5734**	**0.5782**	**0.5867**	**0.6014**
Function-level localization					
None	0.2906	0.2906	0.2906	0.2906	0.2906
Skywork-Reward-Llama-3.1-8B	0.3073	0.2917	0.2663	0.3133	0.2990
Qwen2.5-Coder-7B-Instruct	0.2936	0.3115	0.2775	0.2775	0.3014
HiLoRM(Ours)	**0.3110**	**0.3137**	**0.3175**	**0.3283**	**0.3443**
Line-level localization					
None	0.1361	0.1361	0.1361	**0.1361**	**0.1361**
Skywork-Reward-Llama-3.1-8B	0.1207	0.1168	0.0964	0.1127	0.0902
Qwen2.5-Coder-7B-Instruct	0.1468	0.1089	0.1047	0.1118	0.0781
HiLoRM(Ours)	**0.1732**	**0.1479**	**0.1482**	0.1141	0.0972

D Prompt

The prompt template utilized by the reward model HiLoRM is illustrated in Example 1. This template is consistently applied across all evaluation tasks, regardless of their granularity. Within this template, the `task_definition_and_context` placeholder is designated to hold the task definition and relevant contextual information, while the `candidate_answers` placeholder is used to list multiple candidate answers. Ultimately, the model adheres to the instructions within this prompt to return a response in JSON format, from which the identifier of the selected candidate answer can be parsed.

Example 1: Prompt for Reward Model

You are tasked with selecting the single best answer from a list of potential solutions provided for a specific code-related task.
You will be given two main sections:
1. Task Definition & Context
This section contains all the necessary information to understand the task you need to evaluate responses for. This includes the original prompt that specifies the requirements and expected output format, the problem description you are addressing, and any relevant context like repository structure or file contents.
2. Candidate Answers
This section contains a list of potential solutions that were generated based on the information in the first section.
Your goal is to carefully read and understand everything presented in the Task Definition & Context section. This is crucial for knowing what constitutes a correct and well-formatted answer. Then, you must evaluate each candidate answer presented in the Candidate Answers section. Select the *single* candidate answer that you believe most accurately and effectively solves the problem *and* strictly adheres to the specific instructions and formatting requested within the Task Definition & Context.
———
TASK DEFINITION & CONTEXT:
{task_definition_and_context}
———
CANDIDATE ANSWERS:
{candidate_answers}
———
Selection Instruction:
Based on your comprehensive evaluation of the **Task Definition & Context** and the **Candidate Answers**, select the single best candidate answer.
Provide your selection in the following **strict JSON format**. Do not include any other text, comments, or explanations outside the JSON object. The value for the "answer" key must be the exact unique identifier (e.g., "candidate 1", "candidate 5", "candidate 28") corresponding to your selected best answer from the **Candidate Answers** list.

```json
{
   "answer": "candidate X"
}
```