SWERANK+: MULTILINGUAL, MULTI-TURN CODE RANKING FOR SOFTWARE ISSUE LOCALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Maintaining large-scale, multilingual codebases hinges on accurately localizing issues, which requires mapping natural-language error descriptions to the relevant functions that need to be modified. However, existing ranking approaches are often Python-centric and perform a single-pass search over the codebase. This work introduces SWERANK+, a framework that couples SWERANKMULTI, a cross-lingual code ranking tool, with SWERANKAGENT, an agentic search setup, for iterative, multi-turn reasoning over the code repository. SWERANKMULTI comprises a code embedding retriever and a listwise LLM reranker, and is trained using a carefully curated large-scale issue localization dataset spanning multiple popular programming languages. SWERANKAGENT adopts an agentic search loop that moves beyond single-shot localization with a memory buffer to reason and accumulate relevant localization candidates over multiple turns. Our experiments on issue localization benchmarks spanning various languages demonstrate new state-of-the-art performance with SWERANKMULTI, while SWERANKAGENT further improves localization over single-pass ranking.

1 INTRODUCTION

The maintenance of large-scale software systems constitutes a significant and ever-growing portion of the software development lifecycle. A persistent bottleneck in this process is *software issue localization* Wong et al. (2016): the task of identifying where in a codebase a fix should be applied for a given bug report or feature request. This task requires mapping natural language descriptions, such as those found in GitHub issues, to specific code elements including files, modules, or functions. As modern code repositories grow in size and complexity to encompass thousands of files across multiple programming languages, manual localization becomes increasingly infeasible. Automating this process can therefore accelerate issue resolution and significantly enhance developer productivity.

Recent advances in large language models (LLMs) have led to the development of sophisticated *agentic systems* Yu et al. (2025); Chen et al. (2025) capable of navigating complex codebases. These systems operate by issuing commands to read files, search for patterns, and analyze dependencies. While powerful, such approaches often suffer from high latency and substantial computational overhead. To improve efficiency, SWERANK Reddy et al. (2025) reformulates issue localization as a *retrieve-and-rerank* problem, leveraging specialized bi-encoder retrievers and LLM rerankers trained on curated datasets Suresh et al. (2024). This approach has achieved high precision while substantially reducing computational cost.

Despite its success, SWERANK faces two key limitations that reflect the evolving needs of modern software engineering. First, its scope is largely confined to Python-dominated repositories, whereas real-world enterprise systems are inherently *multilingual*, comprising interconnected components written in diverse programming languages. A practical localization tool must therefore generalize across languages. Second, SWERANK's single-pass ranking design may be inadequate for complex issues that demand *iterative reasoning* or involve changes dispersed across multiple, loosely coupled functions. To overcome these challenges, we introduce SWERANK+, a comprehensive framework for multilingual, multi-turn issue localization.

Our first component, SWERANKMULTI, extends SWERANK to a multilingual setting. It includes a suite of retriever and reranker models trained on SWELOCMULTI, a newly curated, large-scale dataset containing high-quality issue–code pairs spanning multiple popular programming languages.

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.

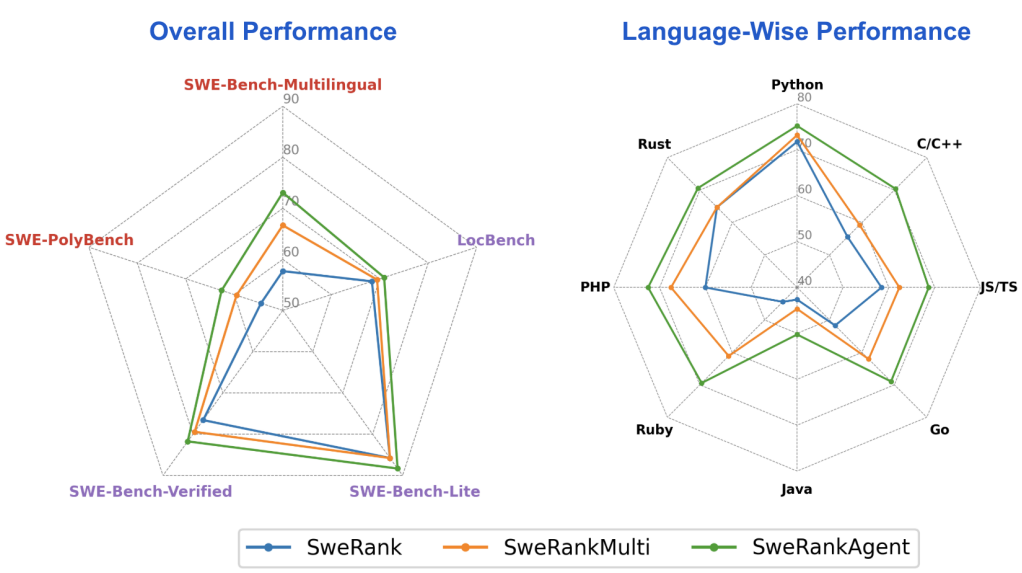


Figure 1: Comparison of function localization accuracy@10 against the SWERANK baseline. SWERANKMULTI shows significant improvement on multilingual benchmarks (in red) while maintaining strong performance on Python-specific evaluations (in violet). SWERANKAGENT further improves over single-pass ranking approaches across the board.

SWERANKMULTI enables high-accuracy localization across heterogeneous repositories, marking the first such effort in this domain.

Next, we present SWERANKAGENT, a lightweight agentic framework that moves beyond single-pass localization. Instead of relying on a single retrieval pass, the agent adopts an iterative process: it begins with an initial hypothesis and progressively refines its understanding through multiple turns. Each turn allows it to gather new evidence, narrow the search space, and focus on the most relevant code regions. This multi-turn reasoning process mirrors how human developers investigate issues–starting with a broad, symptom-based exploration and converging toward the root cause. Specifically, SWERANKAGENT is motivated by three key observations: (A) **Initial searches can be misleading**–the most obvious function in a traceback is often not the true source of the bug; (B) **Context is built over turns**–early retrievals reveal important structural clues about the codebase; (C) **Iterative refinement leads to precision**–repeated reasoning helps transition from symptom-level to root-cause localization.

SWERANKAGENT integrates SWERANKMULTI as a specialized retrieval tool within this iterative process. It can issue multiple search queries, maintain a memory buffer of intermediate results, and reason over aggregated evidence to produce more accurate localizations. This hybrid design combines the efficiency of high-quality retrievers with the depth of agentic reasoning, enabling it to solve complex localization problems that remain intractable under single-pass methods.

Through extensive experiments, we show that SWERANKMULTI establishes new state-of-the-art performance for issue localization across multiple programming languages Rashid et al. (2025); Zan et al. (2025); Yang et al. (2025b), while maintaining competitive results on Python benchmarks Jimenez et al. (2024); Chen et al. (2025). Furthermore, the iterative reasoning in SWERANKAGENT consistently outperforms single-pass ranking for issue localization. Figure 1 summarizes our experimental results. Our contributions are:

- We present the first framework to address issue localization in a multilingual setting.
- We introduce SWERANKMULTI, trained on 10 programming languages, achieving state-of-the-art multilingual code ranking performance for issue localization.
- We propose SWERANKAGENT, an iterative, multi-turn localization framework that further improves over single-pass ranking.

2 Related Work

2.1 Software Issue Localization

Software issue localization (fault localization) aims to identify the specific code regions responsible for software defects. Traditional approaches include spectrum-based methods (Jones & Harrold, 2005) and state-based methods (Zeller, 2002), which analyze program executions and state differences to isolate potential fault sites. Learning-based methods such as DeepFL (Li et al., 2019) extended these ideas by integrating multiple fault signals using deep representations. Recent approaches to fault localization Torun et al. (2025); Chang et al. (2025); Yaraghi et al. (2025) have leveraged LLM-based techniques by incorporating semantic reasoning and retrieval-augmented inference, moving beyond traditional spectrum- and mutation-based metrics.

The advances in AI for software engineering have spurred the development of LLM-based agentic frameworks designed to perform complex software engineering tasks (He et al., 2025; Dong et al., 2025). Recent LLM-based agentic approaches tackles issue localization as a planning and searching problem. Yang et al. (2024a) proposed LLM-based framework that leverages multi-turn reasoning and tool invocation for issue resolution and code repair. Chen et al. (2025) incorporates graph-guided reasoning for precise function localization, while Yu et al. (2025) designs a framework specialized for bug localization using search and read actions.

While these approaches demonstrate the power of agentic architectures, they often involve complex interactions with the codebase, which can incur high latency (Chen et al., 2025; Yu et al., 2025). SweRank+ introduces a distinct, hybrid agentic model, SweRankAgent, that combines the efficiency of a specialized retrieval tool with a lightweight, iterative reasoning loop. This enables tackling complex issues that are intractable for single-pass ranking systems while avoiding the high overhead of complex agentic frameworks.

2.2 Multilingual Code Understanding

Recent code LLMs have significantly expanded their code understanding capabilities across programming languages, with models such as Qwen3-Coder (Yang et al., 2025a), and CodeGemma (Team et al., 2024) exhibiting strong zero-shot transfer between languages. However, the integration of multilinguality into software issue localization remains comparatively nascent. Most localization systems remain monolingual, often trained exclusively on Python (e.g., SweRank (Reddy et al., 2025), LocAgent (Chen et al., 2025)), limiting their applicability to real-world software ecosystems. SweRank+ bridges this gap as the first localization framework explicitly trained and evaluated on multilingual repositories.

3 SweRankMulti

We introduce SweRankMulti, a framework for effective software issue localization across diverse programming languages. SweRankMulti adapts the efficient retrieve-and-rerank methodology of SweRank to the multilingual setting, comprising two key components: the SweRankEmbed-Multi retriever (§3.2), which pre-selects a small set of candidate functions from large codebases most likely relevant to a given issue, and the SweRankLLMMulti reranker (§3.3), which produces a refined final ranking from the top candidates.

3.1 SweLocMulti Training Dataset

We create SweLocMulti, a large-scale multilingual dataset curated specifically for issue localization. While SweLoc Reddy et al. (2025) provides a high-quality resource for Python, modern software systems are inherently multilingual. SweLocMulti extends the data collection and filtering pipeline of SweLoc to encompass JavaScript, Java, TypeScript, Ruby, Rust, Go, PHP, C, and C++. Table 1 provides the detailed language-wise distribution of training instances. We ensure that none of the repositories in the training set overlap with those in the evaluation sets detailed in §5.1.

Issue Collection: Following SWERANK's methodology, we identify popular open-source repositories on GitHub for each language, filtering for repositories with at least 40% code in the target language, over 1,000 stars, and at least one commit in the preceding six months. From this curated set, we extract pull requests (PRs) explicitly linked to GitHub issues that include test file modifications.

Language	# Repos	# PRs	# Instances
JavaScript	104	1513	4254
Java	130	5518	19239
TypeScript	129	3882	11410
Ruby	308	4244	9048
Rust	269	5879	22255
Go	114	2985	11242
PHP	206	4591	16608
C	74	1023	4013
C++	278	2359	7621
Python	2448	24285	49973
Total	4060	56279	155663

Table 1: Distribution of repositories, pull requests (PRs), and training instances across different programming languages in the SWELOCMULTI dataset.

Contrastive Data: Each issue description serves as a query, with modified functions in the corresponding PR treated as positive examples. To ensure the model learns to distinguish fine-grained semantic differences, we employ *consistency filtering* and *hard-negative mining* techniques Suresh et al. (2024). Specifically, we use a pretrained embedding model (SWERANKEMBED-Small) to perform consistency filtering, retaining only examples where the positive function ranks within the top-40 semantically similar functions repository-wide. We then mine hard negatives, which are unmodified functions from the same repository that are semantically similar to the query, creating challenging instances for the contrastive training process.

3.2 SWERANKEMBEDMULTI RETRIEVER

The retriever component, SWERANKEMBEDMULTI, is a bi-encoder model that maps issue descriptions and code functions to dense vector representations in a shared multilingual embedding space. We initialize the model with Qwen3-Embedding Zhang et al. (2025), a state-of-the-art text embedding model, and finetune it on SWELOCMULTI using the InfoNCE contrastive loss Oord et al. (2018), identical to SWERANKEMBED's training objective. Given an issue description as the query, the loss function encourages the query embeddings to be closer to that of the corresponding positive code function while pushing away from other functions in the training batch, including mined hard negatives. During inference, all the repository functions are ranked by cosine similarity between their embeddings and the issue embedding, to identify the most probable fix locations.

3.3 SWERANKLLMMULTI RERANKER

The reranker component, SWERANKLLMMULTI, refines the initial candidate list from the retriever using listwise reranking. This approach leverages the instruction-following capabilities of large language models and has proven more effective than pointwise Zhuang et al. (2023; 2024) or pairwise ranking Qin et al. (2024).

Training SWERANKLLMMULTI leverages the novel weakly supervised strategy introduced in SWERANK, which enables listwise reranking learning even when only a single positive example is known, without requiring a fully ordered ground-truth list. Each candidate function receives a unique identifier, and the model is trained to generate the identifier of the true positive function as its first token using standard language modeling loss. This objective effectively aligns model generation with ranking behavior, teaching it to select the most relevant function from the candidate set. Fine-tuning on the multilingual SWELOCMULTI dataset substantially enhances cross-language generalization and code ranking accuracy, setting a new state of the art for multilingual issue localization (§5.2).

4 SWERANKAGENT

While SWERANKMULTI models effectively localize relevant code functions, they operate on a single, static query–the original issue description. This approach performs well for self-contained issues where the initial report provides sufficient signal to identify the fix location. However, many software maintenance tasks involve complex bugs or feature requests whose critical details are not fully captured in the initial description. For example, a bug report may describe only a high-level

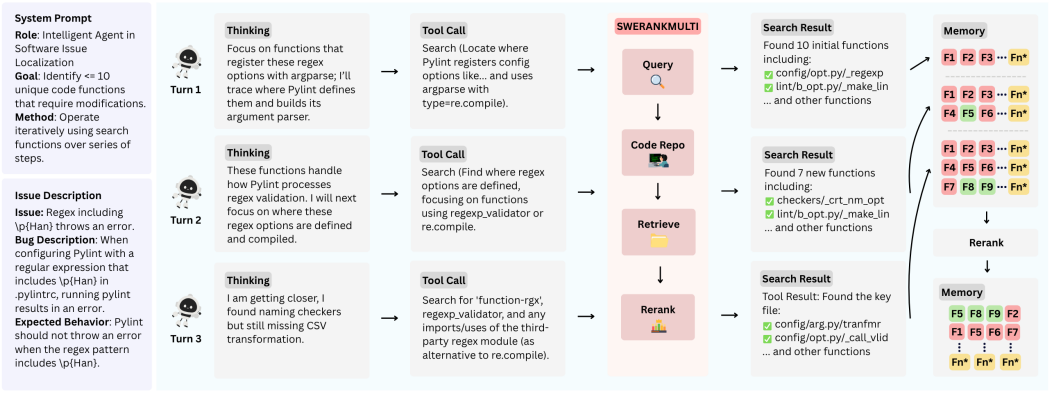


Figure 2: SWERANKAGENT interleaves *Search* actions, which retrieve candidate functions, and *Aggregate* actions, which selectively adds the most relevant functions to a persistent memory. The diagram illustrates a qualitative example of the agent solving a complex issue. The agent's multi-turn trajectory (turns 1-3) demonstrates how it iterates through thinking and tool calls to SWERANKMULTI to progressively narrow down the root cause from a broad search to the target function.

symptom, and it is only after examining the functions related to that symptom that a developer can pinpoint the downstream component responsible for the fault.

To emulate this human-like, iterative exploratory process, we introduce SWERANKAGENT, a lightweight and generic agentic framework that extends SWERANKMULTI with multi-turn search capabilities. Rather than treating issue localization as a single-shot ranking task, SWERANKAGENT performs iterative searches across multiple turns to progressively accumulate and refine localization evidence. This design enables the agent to decompose complex localization problems into a sequence of smaller, more tractable steps, effectively overcoming the limitations of single-pass retrieval.

SWERANKAGENT operates as a ReAct-style Yao et al. (2023) reasoning agent, alternating between reasoning and action-taking. Its iterative operational loop, illustrated in Figure 2, consists of four key steps: `Search`, `Reasoning`, `Reformulation`, and `Aggregate`.

- `Search` with *issue description*: The agent's primary interface with the codebase. Given an issue description as the query, this action invokes the SWERANKMULTI retriever and reranker as a tool to obtain the top-k most relevant functions from the code repository.
- `Reasoning` over *retrieved functions*: The agent analyzes the retrieved functions to assess their relevance to the issue and determine whether further exploration is needed. This step guides subsequent query refinement.
- `Reformulation` for *query update*: After the reasoning step, the agent updates the query based on the previous search and its assessment of the retrieved functions' relevance. This enables the agent to progressively refine its search and improve contextual understanding across iterations.
- `Aggregate` on *candidate functions*: Across turns, the agent accumulates identifiers of relevant functions in an internal memory buffer that serves as a global candidate pool. This enables the agent to capture issues whose fixes span multiple functions that may not co-occur within a single retrieval. Finally, the SWERANKLLMMULTI reranker is applied to the aggregated pool to produce a globally ranked list of candidate functions.

The agent iteratively cycles through `Search`, `Reasoning`, and `Reformulation` until a stopping condition is met–either the maximum number of iterations is reached or no new relevant functions are retrieved. The accumulated candidates are then consolidated and reranked during the final `Aggregate` step, with the top-10 functions returned as the final localization output. Throughout this process, the underlying LLM guides the agent's reasoning, enabling it to refine queries, evaluate search results, and dynamically adapt its strategy based on evolving evidence from the codebase.

Model	Multilingual				Python					
	SWE-PolyBench		SWE-Bench-Multilingual		SWE-Bench-Lite		LocBench		SWE-Bench-Verified	
	Acc@5	Acc@10	Acc@5	Acc@10	Acc@5	Acc@10	Acc@5	Acc@10	Acc@5	Acc@10
OpenHands (Claude-3.5)	-	-	-	-	68.25	70.07	-	59.11	-	-
LocAgent (Claude-3.5)	-	-	-	-	73.36	77.37	-	59.29	-	-
Gemini-Embedding (unknown)	-	-	36.75	47.44	61.31	72.26	43.04	51.43	59.74	66.96
CodeRankEmbed (137M)	27.11	33.40	26.50	35.04	51.82	58.76	38.93	47.86	50.98	57.99
Qwen3-Embedding-0.6B	30.69	37.17	30.34	38.03	52.55	62.77	39.64	47.32	49.45	60.39
SWERANKEMBED-Small (137M)	35.91	42.79	33.33	44.02	63.14	74.45	51.79	58.57	59.74	68.49
SWERANKEMBEDPYTHON-Small (0.6B)	37.75	43.95	36.75	47.01	**66.79**	75.18	50.00	58.04	61.93	71.12
SWERANKEMBEDMULTI-Small (0.6B)	**39.01**	**47.14**	**43.16**	**52.56**	**66.79**	**76.28**	**51.25**	**58.93**	**64.99**	**71.77**
GTE-Qwen2-7B-Instruct (7B)	33.40	39.40	34.19	42.31	63.14	70.44	42.50	51.79	57.77	65.21
Qwen3-Embedding-8B	39.50	46.47	36.75	46.15	60.95	71.53	44.46	55.00	60.18	65.21
SWERANKEMBED-Large (7B)	41.92	49.18	39.74	50.85	71.90	82.12	55.18	63.21	65.65	74.18
SWERANKEMBEDPYTHON-Large (8B)	44.24	51.98	46.15	55.98	73.72	83.94	55.71	65.00	66.96	75.05
SWERANKEMBEDMULTI-Large (8B)	**46.56**	**53.73**	**50.43**	**62.39**	**77.37**	**86.86**	**56.43**	**65.36**	**68.71**	**76.37**

Table 2: Function localization performance of different retrievers in comparable sizes across benchmarks with various programming languages. Our SWERANKEMBEDMULTI models considerably improve performance on both multilingual and python-specific benchmarks.

5 EXPERIMENTS

Our experiments are designed to address the following research questions: **RQ1:** *How effective is training on* SWELOCMULTI *compared to the Python-specific* SWELOC? and **RQ2:** *Can issue localization benefit from multi-turn search*?. To examine **RQ1**, Section §5.2 reports the performance of the SWERANKMULTI retriever and reranker across multilingual and Python-specific issue localization benchmarks. To investigate **RQ2**, Section §5.3 analyzes the effectiveness of multi-turn issue localization using SWERANKAGENT compared to the single-turn SWERANKMULTI.

5.1 DATASETS & METRICS

Our multilingual evaluation data comprises two datasets: SWE-PolyBench Rashid et al. (2025) and SWE-Bench-Multilingual Yang et al. (2025b). Each (PR, codebase) instance pair from these datasets is transformed into the localization format, where the PR's corresponding github issue description serves as the retrieval query. The Tree-sitter parsing tool is employed to extract all functions from the codebase, creating candidate corpus. Functions modified within the PR are considered as positives. We also include python-specific benchmarks, specifically SWE-Bench-Lite Jimenez et al. (2024), LocBench Chen et al. (2025) and SWE-Bench-Verified Chowdhury et al. (2024). For evaluation, we use Accuracy at k (Acc@k), which deems localization successful if all relevant code locations are identified within the top-k results. This recall-oriented measure better captures the benefit of localization for downstream code-repair models, as opposed to ranking-oriented measures like MRR.

5.2 SWERANKMULTI

5.2.1 SETUP

Model Training: We train the SWERANKEMBED model in two sizes: *small* and *large*. Both models are trained on the SWELOCMULTI dataset (§3.1), with small and large variants initialized with the 0.6B and 8B variants of Qwen3-Embedding (Zhang et al., 2025) respectively. The SWERANKEMBEDMULTI-SMALL encoder is finetuned for two epochs with a learning rate of 2e-5, a batch size of 64 and 15 hard negatives per example. The SWERANKEMBEDMULTI-LARGE encoder is finetuned for 1 epoch with a learning rate of 8e-6, a batch size of 64 and 7 hard negatives per example. Both models are finetuned using 8 GH200 GPUs. Our data filtering, negative mining, and model finetuning are implemented using the contrastors package Nussbaum et al. (2024).

Following SWERANK Reddy et al. (2025), the SWERANKLLMMULTI small and large rerankers are initialized with CodeRankLLM (7B) Suresh et al. (2024) and Qwen-2.5-32B-Instruct Yang et al. (2024b) respectively, and finetuned with SWELOCMULTI for listwise reranking. For the LLM reranker training, we trained for one epoch with a global batch size of 128, an initial learning rate of 5e-6 with 50 warmup steps, cosine learning rate scheduler, bfloat16 precision, and noisy embeddings Jain et al. (2023) with a noise scale $\alpha = 5$. For efficient long-context, multi-gpu training,

Model	Multilingual				Python					
	SWE-PolyBench		SWE-Bench-Multilingual		SWE-Bench-Lite		LocBench		SWE-Bench-Verified	
	Acc@5	Acc@10	Acc@5	Acc@10	Acc@5	Acc@10	Acc@5	Acc@10	Acc@5	Acc@10
SweRankEmbedMulti-Small (0.6B)	39.01	47.14	43.16	52.56	66.79	76.28	51.25	58.93	64.99	71.77
+ CodeRankLLM (7B)	42.40	51.11	48.72	58.55	72.99	80.29	55.54	63.93	66.52	75.93
+ Qwen3-Instruct-8B	43.66	51.98	50.85	59.83	72.27	80.29	56.96	64.46	68.49	75.27
+ SweRankLLM-Small (7B)	51.21	56.92	55.13	63.68	77.37	85.04	**63.39**	**69.64**	**73.30**	77.68
+ **SweRankLLMMulti-Small (7B)**	**53.15**	**59.54**	**56.41**	**66.67**	**80.29**	**85.77**	63.04	69.46	**73.30**	**79.43**
SweRankEmbedMulti-Large (8B)	46.56	53.73	50.43	62.39	77.37	86.86	56.43	65.36	68.71	76.37
+ GPT-4.1	55.66	61.76	62.39	70.94	79.93	88.69	65.89	70.89	76.81	**81.62**
+ SweRankLLM-Large (32B)	56.73	62.73	57.69	70.51	83.58	89.42	64.64	71.25	75.49	80.96
+ **SweRankLLMMulti-Large (32B)**	**58.28**	**63.21**	**64.10**	**71.37**	**85.40**	**89.78**	**66.43**	**71.96**	**78.12**	81.18

Table 3: Function localization accuracy of different rerankers on multilingual and python-specific benchmarks.

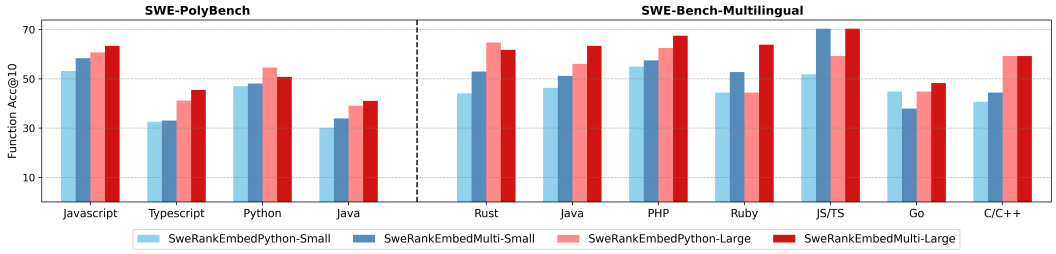


Figure 3: Language-wise localization accuracy for SWE-PolyBench and SWE-Bench-Multilingual.

we used DeepSpeed Rasley et al. (2020) ZeRO stage 3 with 16 GH200 GPUs. To prevent the positional bias from affecting the reranker and ensure model robustness Pradeep et al. (2023), we shuffle the order of candidate codes for each training example.

Baselines: For the retriever evaluation, we compare against existing code embedding models such SweRankEmbed (Reddy et al., 2025), and CodeRankEmbed (Suresh et al., 2024). Since SweRankEmbed was finetuned on GTE-Qwen2-7B-Instruct, we finetune the Qwen3-Embedding models on the python-specific SweLoc dataset (Reddy et al., 2025) to get the SweRankEmbedPython variants for better comparison. We also include Gemini-Embedding (Lee et al., 2025), the current top model on the MTEB leaderboard, as a general-purpose closed-source baseline. For the reranker evaluation, we compare against listwise reranker models such as SweRankLLM Reddy et al. (2025), CodeRankLLM Suresh et al. (2024), and the zero-shot Qwen3-Instruct-8B model Yang et al. (2025a) along with GPT-4.1.

5.2.2 Results

Table 2 shows the function localization performance different embedding models when compared at similar size ranges. We see that SweRankEmbedMulti achieves SOTA performance at both size variants, considerably improving upon the existing SweRankEmbed. Moreover, we see that a better base embedding model (SweRankEmbed vs SweRankEmbedPython) and training with multilingual data (SweRankEmbedPython vs SweRankEmbedMulti) both contribute to improvements in performance. Interestingly, SweRankEmbedMulti even improves over SweRankEmbedPython on python-specific benchmarks, despite the latter being trained on the python-exclusive SweLoc. This could be attributed to better generalization performance from including more languages in the training data.

Table 3 shows the function localization performance of the reranker when used with retrievers of different sizes. We see that SweRankLLMMulti reranker consistently improve localization performance over the SweRankLLM models, with the large variant even outperforming GPT-4.1.

Language-wise Performance: Figure 3 shows performance of the retrievers separately for each of the languages in SWE-PolyBench and SWE-Bench-Multilingual. We observe that both the small and large variants of SweRankEmbed consistently improve performance on most languages, compared to SweRankEmbedPython.

Approach	Multilingual		Python		
	SWE-PolyBench	SWE-Bench-Multilingual	SWE-Bench-Lite	LocBench	SWE-Bench-Verified
SWERANKMULTI (Single-Query)	59.54	66.67	85.77	69.46	79.43
Reformulate (Single-Turn, Multi-Query)	58.08 (-1.46)	64.96 (-1.71)	85.77	69.86 (+0.40)	78.99 (-0.44)
SWERANKAGENT (Multi-Turn)	**62.63** (+3.09)	**73.08** (+6.41)	**88.32** (+2.55)	**70.91** (+1.45)	**81.74** (+2.31)

Table 4: The multi-turn SWERANKAGENT consistently outperforms both the single-turn SWERANKMULTI baseline and a single-turn, multi-query reformulation approach on function localization performance (Acc@10) across Multilingual and Python benchmarks.

5.3 SWERANKAGENT

5.3.1 SETUP

SWERANKMULTI tool: We equip the SWERANKAGENT framework with a search tool that, when given with a query in the form of an issue description, returns the top-10 localized functions within the codebase. The SWERANKMULTIEMBED retriever first obtains the top-100 results which are then passed to the SWERANKMULTILLM reranker. For efficiency considerations, the codebase function embeddings for the retriever are pre-cached. Given compute considerations, we only use the small variants of the retriever and reranker for the SWERANKMULTI tool.

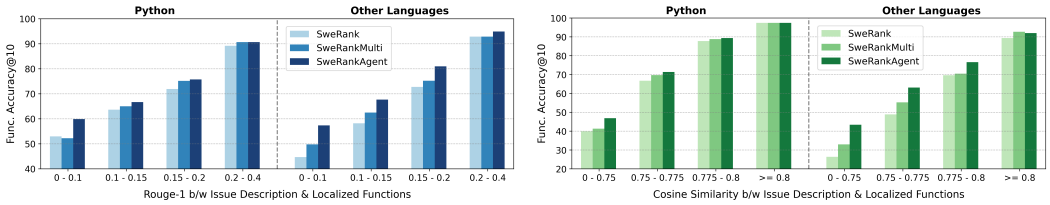
Baselines: To investigate the performance of a multi-query setup, we design a *Reformulate* baseline that does query-reformulation to generate multiple queries from the original issue description. In total, we use the original issue description as the query plus five reformulated queries generated by GPT-5. For each query, results from the SWERANKMULTI tool are collected, and the aggregated output is then again passed to the SWERANKMULTILLM reranker to obtain final top-10 function localization result. Moreover, we also use the SWERANKMULTI tool as a single-query baseline.

5.3.2 RESULTS

Table 4 shows the function localization performance of SWERANKAGENT using GPT-5 as the underlying LLM. Firstly, we observe that the multi-query *Reformulate* baseline yields lower performance than the single-query baseline. This suggests that directly using reformulated queries introduces additional noise, highlighting the inherent challenges in improving coverage of localization candidates with multiple queries. In contrast, SWERANKAGENT consistently improves localization performance across all datasets, with more than a 3-point boost on SWE-PolyBench and over a 6-point boost on SWE-Bench-Multilingual.

Qualitative Example: Figure 2 illustrates the advantage of multi-turn issue localization through a qualitative example where iterative search succeeds while single-pass ranking fails. At first glance, the issue appears straightforward: the pylint *re.error* traceback points to *re.compile*. Single-pass ranking would likely point at *_regexp_validator*, the immediate crash site. However, the real bug lies upstream, in how a comma-separated list of regular expressions is parsed before validation. Modifying *_regexp_validator* would thus be incorrect; the fix belongs in the function handling the comma-separated-value (CSV) input. SWERANKAGENT's multi-turn trajectory below demonstrates how iterative refinement enables correct localization.

- **Turn 1: Broad Exploration.** The agent begins with a broad query to locate where configuration options are defined and validated. This identifies relevant configuration modules in *arguments_manager.py* and the symptom function *_regexp_validator*, but not the root cause. A single-pass approach would likely stop here.
- **Turn 2: Contextual Refinement.** Based on the initial results, the agent infers that the issue relates to how naming options are validated, refining its search to focus on functions involved in regular expression validation. This uncovers key functions such as *_regexp_csv_validator*, revealing the CSV-handling pathway.
- **Turn 3: Pinpointing the Target.** The results point to a transformer function that processes CSV inputs prior to validation. A focused search on transformer–validator interactions returns the correct target: *_regexp_csv_transformer*.



(a) Performance breakdown by lexical overlap.

(b) Performance breakdown by semantic overlap.

Figure 4: SWERANK, SWERANKMULTI, and SWERANKAGENT performance across across different difficulty tiers. Difficulty is measured by (a) lexical overlap (ROUGE-1) and (b) semantic overlap (cosine similarity) between the issue description and the true localized functions.

5.3.3 ANALYSIS BY ISSUE COMPLEXITY

Aggregate performance metrics can mask how localization methods behave under varying levels of problem difficulty. We thereby analyze performance across two orthogonal dimensions of complexity: (1) the degree of lexical and semantic overlap between the issue description and target code, and (2) the number of ground-truth localized functions. We primarily aim to investigate whether the agentic search process is particularly beneficial for harder instances with multiple target functions or requiring matching beyond surface-level similarity.

Performance by Overlap: We bucket instances based on lexical overlap (ROUGE-1) and semantic overlap (cosine similarity) between the issue description and the ground-truth localized functions, with results shown in Figure 4. We can see that performance generally improves with higher overlap, indicating that keyword- or semantically-aligned issues are easier to localize. Nonetheless, SWERANKAGENT demonstrates clear advantages in low-overlap settings. In buckets with minimal lexical or semantic overlap, where naive keyword matching is insufficient, the agent consistently shows bigger improvements compared to buckets with more overlap. The agent's ability to reformulate queries and reason over intermediate results allows it to bridge the gap when the issue description lacks specific keywords present in the target functions. Hence, multi-turn reasoning is crucial for traversing the "semantic gap" in harder instances where the failure description is distant from the root cause.

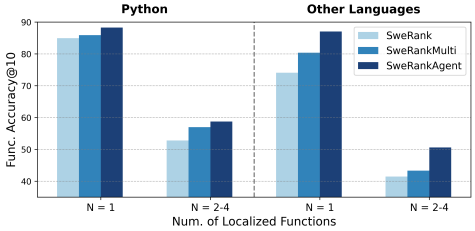


Figure 5: Function Accuracy@10 breakdown by number of target localization functions.

Performance by # of Target Functions: We stratify test instances by the number of functions modified in the ground-truth patch, which serves as a proxy for issue complexity. As seen in Figure 5, localization accuracy degrades as the number of target localization functions increases, reflecting the increased difficulty of multi-function bugs and feature requests. While the multilingual training of SWERANKMULTI does considerably improve performance for other languages, SWERANKMULTI particularly shines for more complex multi-function localization settings.

6 CONCLUSION

In this work, we introduced SWERANK+, a comprehensive software issue localization framework. Our contributions are twofold. First, we developed SWERANKMULTI, a multilingual retrieve-and-rerank system trained on the newly curated SWELOCMULTI dataset, that establishes new state-of-the-art performance on multilingual benchmarks while maintaining strong efficacy on Python-specific tasks. Second, we proposed SWERANKAGENT, a lightweight agentic framework that employs iterative, multi-turn reasoning to refine localization candidates, which consistently outperforms single-pass ranking. Our analysis highlights that this agentic approach is particularly effective for complex issues characterized by low overlap between issue description and target functions or those requiring modifications across multiple functions. Our work emphasizes a hybrid design that bridges the gap between efficient retrieval systems and computationally heavy agentic frameworks, demonstrating that lightweight, iterative reasoning can effectively solve complex issue localization.

AI USE STATEMENT

Generative AI tools were employed strictly in an assistive capacity during this project. Specifically, we used AI-assisted development environments (namely Cursor) to aid in implementation, script generation for analysis plots, and literature discovery for the software issue localization related work subsection. Generative AI was used for rewriting / rephrasing certain content but not used for conceptual ideation or for drafting manuscript text from scratch.

The authors have thoroughly verified all AI-assisted outputs: citations and literature recommendations were manually confirmed for authenticity, and generated plotting scripts and figures were validated against raw underlying data. The authors take full responsibility for all content, claims, and artifacts presented in this paper.

REFERENCES

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

Zhaoling Chen, Xiangru Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. Locagent: Graph-guided llm agents for code localization. *arXiv preprint arXiv:2503.09089*, 2025.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. A survey on code generation with llm-based agents, 2025. URL https://arxiv.org/abs/2508.00083.

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*, 34(5):1–30, 2025.

Neel Jain, Ping yeh Chiang, Yuxin Wen, John Kirchenbauer, Hong-Min Chu, Gowthami Somepalli, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Neftune: Noisy embeddings improve instruction finetuning, 2023. URL https://arxiv.org/abs/2310.05914.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

James A Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pp. 273–282, 2005.

Jinhyuk Lee, Feiyang Chen, Sahil Dua, Daniel Cer, Madhuri Shanbhogue, Iftekhar Naim, Gustavo Hernández Ábrego, Zhe Li, Kaifeng Chen, Henrique Schechter Vera, et al. Gemini embedding: Generalizable embeddings from gemini. *arXiv preprint arXiv:2503.07891*, 2025.

Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization. In *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*, pp. 169–180, 2019.

Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. Nomic embed: Training a reproducible long context text embedder, 2024.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. In *Advances in Neural Information Processing Systems*, pp. 10203–10213, 2018.

Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. Rankvicuna: Zero-shot listwise document reranking with open-source large language models, 2023. URL https://arxiv.org/abs/2309.15088.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, et al. Large language models are effective text rankers with pairwise ranking prompting. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 1504–1518, 2024.

Muhammad Shihab Rashid, Christian Bock, Zhuang Yuan, Alexander Buccholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, Anoop Deoras, Giovanni Zappella, and Laurent Callot. Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents. *ArXiv*, abs/2504.08703, 2025. URL https://api.semanticscholar.org/CorpusID:277741387.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3505–3506, 2020.

Revanth Gangi Reddy, Tarun Suresh, JaeHyeok Doo, Ye Liu, Xuan Phi Nguyen, Yingbo Zhou, Semih Yavuz, Caiming Xiong, Heng Ji, and Shafiq Joty. Swerank: Software issue localization with code ranking. *arXiv preprint arXiv:2505.07849*, 2025.

Tarun Suresh, Revanth Gangi Reddy, Yifei Xu, Zach Nussbaum, Andriy Mulyar, Brandon Duderstadt, and Heng Ji. Cornstack: High-quality contrastive data for better code ranking. *arXiv preprint arXiv:2412.01007*, 2024.

CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A. Choquette-Choo, Jingyue Shen, Joe Kelley, Kshitij Bansal, Luke Vilnis, Mateo Wirth, Paul Michel, Peter Choy, Pratik Joshi, Ravin Kumar, Sarmad Hashmi, Shubham Agrawal, Zhitao Gong, Jane Fine, Tris Warkentin, Ale Jakse Hartman, Bin Ni, Kathy Korevec, Kelly Schaefer, and Scott Huffman. Codegemma: Open code models based on gemma, 2024. URL https://arxiv.org/abs/2406.11409.

Utku Boran Torun, Mehmet Taha Demircan, Mahmut Furkan Gön, and Eray Tüzün. Past, present, and future of bug tracking in the generative ai era. *arXiv preprint arXiv:2510.08005*, 2025.

W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016.

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*, 2025a.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024a.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025b. URL https://arxiv.org/abs/2504.21798.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, Shanghaoran Quan, and Zekun Wang. Qwen2.5 technical report. *ArXiv*, abs/2412.15115, 2024b. URL https://api.semanticscholar.org/CorpusID:274859421.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Ahmadreza Saboor Yaraghi, Golnaz Gharachorlu, Sakina Fatima, Lionel C Briand, Ruiyuan Wan, and Ruifeng Gao. Black-box test code fault localization driven by large language models and execution estimation. *arXiv preprint arXiv:2506.19045*, 2025.

Zhongming Yu, Hejia Zhang, Yujie Zhao, Hanxian Huang, Matrix Yao, Ke Ding, and Jishen Zhao. Orcaloca: An llm agent framework for software issue localization. *arXiv preprint arXiv:2502.00350*, 2025.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Kai Shen, and Liang Xiang. Multi-swe-bench: A multilingual benchmark for issue resolving. *ArXiv*, abs/2504.02605, 2025. URL `https://api.semanticscholar.org/CorpusID:277510042`.

Andreas Zeller. Isolating cause-effect chains from computer programs. *ACM SIGSOFT Software Engineering Notes*, 27(6):1–10, 2002.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. Rankt5: Fine-tuning t5 for text ranking with ranking losses. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2308–2313, 2023.

Honglei Zhuang, Zhen Qin, Kai Hui, Junru Wu, Le Yan, Xuanhui Wang, and Michael Bendersky. Beyond yes and no: Improving zero-shot llm rankers via scoring fine-grained relevance labels. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pp. 358–370, 2024.

A PROMPT DETAILS

SWERANKLLMMULTI Prompt

System Prompt

You are CodeRanker, an intelligent code reviewer that can analyze GitHub issues and rank code functions based on their relevance to containing the faults causing the GitHub issue.

User Prompt

I will provide you with 10 code functions, each indicated by a numerical identifier []. Rank the code functions based on their relevance to containing the faults causing the following GitHub issue: <Issue Description>

Code Functions

[1]: <Function 1>
[2]: <Function 2>
...
[10]: <Function 10>

Response Format

All the code functions should be included and listed using identifiers, in descending order of relevance. The output format should be [] ¿ [], e.g., [2] ¿ [1]. Only respond with the ranking results, do not give any explanation.

SWERANKAGENT Prompt

System Prompt

You are an intelligent assistant specializing in software issue localization. Your primary goal is to identify the functions from a given codebase that are most likely to require modification to fix a given issue. You must operate by iteratively using the search tool.

Rules and Guidelines

1. **Iterative Search**: perform sequential 'search' calls; number of rounds is configurable.
2. **Review & Reflect after each 'search'**: use results to inform your next query. Avoid duplicate queries.
3. **Explain & Reformulate**: explain relevance for new functions, then reason about how to refine next query.
4. **Termination**: once coverage is sufficient or rounds are done, call 'finish' with up to 10 functions.

Available Tools

"**name**": "search", "**description**": "Searches the codebase for functions relevant to the query. Returns a list of candidate functions found based on the description of the issue passed to the tool.", "**parameters**": "issue_description"

"**name**": "finish", "**description**": "Call this tool when you are confident you have identified all the top relevant functions.", "**parameters**": null

Expected Response Format

THOUGHT: Summarize what you just learned from the latest search results. For EACH newly added function, provide a brief relevance explanation describing why it may relate to the issue description.

REFORMULATION: Explain how you will adjust the next search query to improve coverage/diversity.

ACTION: {"name": "...", "arguments": { ... }}

User Prompt

< Github Issue Description>