Look Less, Find More: Retrieval-Guided Reinforcement Learning for Agentic Code Localization

Anonymous ACL submission

Abstract

Agentic code localization requires an agent to identify the files, modules, and functions in a repository that must be modified to resolve a natural-language issue. Existing approaches force the agent to both discover and verify relevant locations within a single exploration loop, trading breadth for depth under a fixed turn budget. We present PILOT (**P**re-filtered **I**ssue **L**ocalization with **O**nline **T**rajectories), a two-stage framework that decouples these concerns. A dense retrieval module first selects the top-K function-level candidates and structurally trims the repository via AST rewriting, reducing thousands of functions to K. A 4B-parameter agent trained with Group Sequence Policy Optimization (GSPO) then verifies and refines this focused subset through multi-turn terminal interaction. Across SWE-Bench Verified, Pro, and Lite, PILOT consistently improves function-level F1 by 2.5 to 3.6 pp over the same agent operating on the full repository, with gains at every granularity level. Function-level trimming outperforms file-level trimming despite retaining strictly less content, and the RL-trained agent benefits more from pre-filtering than a vanilla model. The retrieval stage adds negligible cost, requires no changes to the RL training procedure, and reduces median inference time by 8 to 25%, demonstrating that lightweight retrieval and multi-turn reasoning are complementary rather than redundant for repository-scale code localization.

1 Introduction

Repository-level coding tasks such as those in SWE-Bench (Jimenez et al., 2024) require the system to first identify which parts of a large codebase are responsible for an issue before any edit can be proposed. Given a natural-language issue description $\mathcal{I}$ and a pre-edit repository state $\mathcal{R}$, the system must produce a structured prediction $y = (F, M, U)$ consisting of the set of files F, modules M (classes or top-level scopes), and functions U that must be modified. We refer to this task as *issue-conditioned code localization*.

Recent agentic approaches instantiate localization with specialized, language-specific tooling. LocAgent (Chen et al., 2025), OrcaLoca (Yu et al., 2025), and CoSiL (Jiang et al., 2025) build call and dependency graphs using AST-based parsers; RepoSearcher (Ma et al., 2025) introduces retrieval tools that exploit Python imports; RepoNavigator (Zhang et al., 2025) uses a language-server jump tool to resolve symbol definitions. Orthogonally, Sutawika et al. (2026) demonstrate that reinforcement learning over a minimal bash scaffold can match or exceed these specialized systems at compact model scales, eliminating the need for language-specific tooling. Yet regardless of scaffold complexity, all existing approaches share a common bottleneck: the agent must explore the entire repository within a fixed turn budget. As repositories grow to hundreds of files and thousands of functions, this forces the policy to simultaneously discover and verify relevant code locations in a single exploration pass.

We observe that discovery and verification are distinct capabilities with different computational profiles. Discovery (identifying candidate locations) benefits from broad, shallow coverage over the entire codebase and is well suited to lightweight retrieval. Verification (confirming and refining candidates) requires deeper reasoning about code semantics and issue context and is the natural strength of agentic interaction. Conflating the two within a single exploration loop forces the agent to trade breadth for depth under a fixed turn budget.

We present PILOT (**P**re-filtered **I**ssue **L**ocalization with **O**nline **T**rajectories), a two-stage framework that decouples these capabilities. In the first stage, a dense retrieval module encodes function-level code chunks and selects the top-K candidates by embedding similarity,

reducing the function-level search space by 32 to 69× and collapsing files from hundreds to 42 to 56. In the second stage, a GSPO-trained agent verifies and refines this candidate set through terminal-based interaction within a bounded turn budget, operating on a repository that has been structurally trimmed to contain only the retrieved functions and their enclosing scopes. The two stages are complementary in that retrieval provides broad coverage at negligible cost, while the RL-trained agent contributes the semantic reasoning that retrieval alone cannot supply.

We make three contributions.

- A two-stage decomposition of code localization into retrieval-based discovery and RL-based verification, connected by function-level AST trimming that reduces the search space from thousands of functions to K (§3).
- Consistent improvements across all three SWE-Bench splits and granularity levels, with function-level F1 gains of 2.5 to 3.6 pp and 8 to 25% lower inference time, without any changes to RL training (§5).
- Function-level trimming outperforms file-level trimming by 1.4 to 3.2 pp despite retaining strictly less content, and the RL-trained agent benefits more from the reduced search space than a vanilla model, indicating that learned verification behaviors complement retrieval-based filtering (§6.3).

2 Related Work

Code localization. Agentless (Xia et al., 2024) applies hierarchical LLM filtering; LocAgent (Chen et al., 2025), OrcaLoca (Yu et al., 2025), and CoSiL (Jiang et al., 2025) build code graphs via AST parsing; RepoSearcher (Ma et al., 2025) exploits Python imports; and RepoNavigator (Zhang et al., 2025) resolves symbols through a language server. CodeScout (Sutawika et al., 2026) shows that RL over a minimal bash scaffold matches these specialized systems at compact scales. All require the agent to explore the full repository within a fixed turn budget; PILOT removes this requirement by decoupling discovery from verification.

Retrieval-augmented code agents. Dense code embeddings (Husain et al., 2019; Nussbaum et al., 2024) and sparse retrieval (Robertson and Zaragoza, 2009) support code search and retrieval-augmented generation (Wang et al., 2025b). SWE-grep (Pan et al., 2025a) injects retrieved snippets into the agent's prompt. PILOT instead *structurally trims* the repository via AST rewriting, physically removing distractors rather than enriching context.

RL for software-engineering agents. SWE-RL (Wei et al., 2026), DeepSWE (Luo et al., 2025), SWE-Gym (Pan et al., 2025b), R2E-Gym (Jain et al., 2025), and SWE-Smith (Yang et al., 2026) train or provide environments for issue-resolution agents; GSPO (Zheng et al., 2025) and DR.GRPO (Liu et al., 2025) improve multi-turn policy optimization; SkyRL (Cao et al., 2025; Griggs et al., 2025) provides the long-horizon RL framework we build on. These works train agents on full repositories. PILOT is complementary, constraining the search space before the RL agent acts so the policy focuses on verification rather than discovery.

3 Method

Given a GitHub issue $\mathcal{I}$ in a Python repository $\mathcal{R}$, the agent must predict $y = (F, M, U)$, the set of files F, modules M (classes or top-level scopes), and functions U that must be modified. Ground-truth labels $y^\star = (F^\star, M^\star, U^\star)$ are extracted from the issue-resolution patch. PILOT operates in two stages: a dense retrieval module that selects a compact set of candidate functions from the full repository (§3.1), followed by a GSPO-trained agent that verifies and refines this set through terminal-based interaction (§3.2). We detail each stage below and conclude with the GSPO training procedure (§3.3). Figure 1 provides an overview.

3.1 Stage 1. Dense Retrieval Pre-Filtering

The first stage reduces the agent's search space from the full repository to a focused subset of K candidate functions. Let $\mathcal{F} = \{f_1, \ldots, f_N\}$ denote all function and method definitions extracted from every Python file in $\mathcal{R}$ at the instance's base commit. Each function f_i is represented as a document consisting of its fully qualified name (file path, class, and function name) concatenated with its source code.

We encode all function documents using Nomic Embed Code (Nussbaum et al., 2024), a dense code embedding model, with a maximum sequence length of 2,048 tokens. Let ϕ denote the encoder. We compute embeddings $\mathbf{e}_f = \phi(f)$ for each $f \in \mathcal{F}$ and a query embedding $\mathbf{q} = \phi(\mathcal{I})$ for

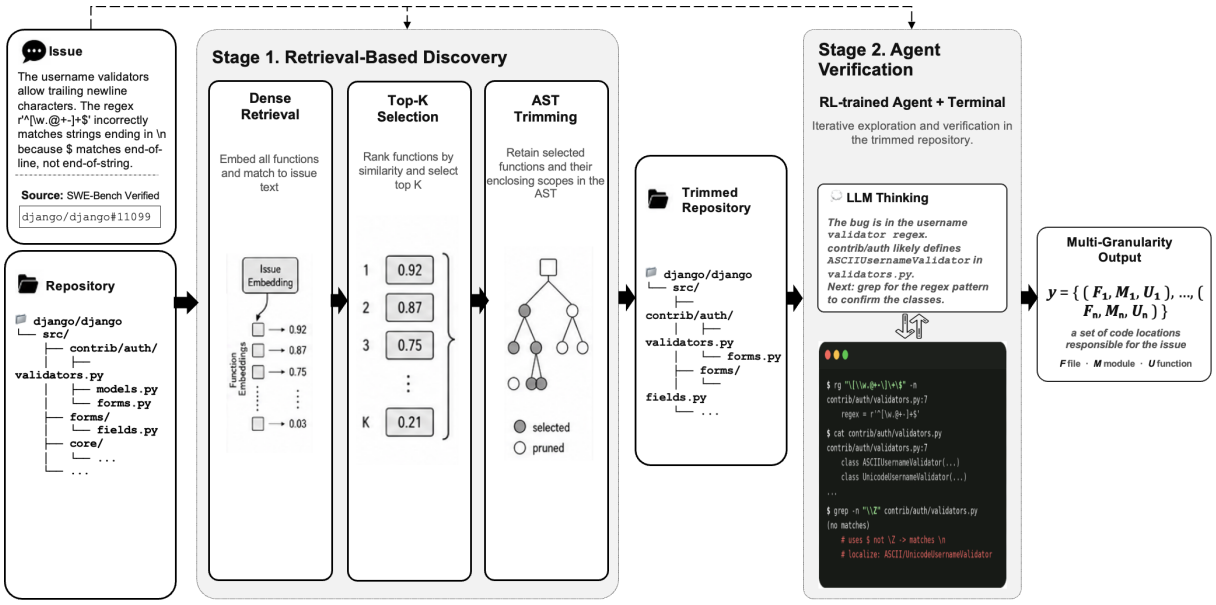

Figure 1: Overview of PILOT. Stage 1 scores each function against the issue using a precomputed embedding index, selects the top-K candidates, and structurally trims the repository via AST rewriting to retain only the selected functions and their enclosing scopes. The embedding index is built once per repository commit and reused across issues. Stage 2 deploys an RL-trained agent that explores the trimmed repository through terminal commands and outputs a multi-granularity prediction of files, modules, and functions to modify.

the issue text. Each function is scored by cosine similarity,

$$s(f, \mathcal{I}) = \frac{\mathbf{e}_f^\top \mathbf{q}}{\|\mathbf{e}_f\| \, \|\mathbf{q}\|}, \tag{1}$$

and the top-K functions form the candidate set

$$C_K = \text{top-K}_{f \in \mathcal{F}} \; s(f, \mathcal{I}). \tag{2}$$

Given C_K, we structurally trim the repository checkout using AST rewriting. For each Python file, we parse the AST and retain only the nodes that are in C_K or are ancestors of a node in C_K. This preserves the enclosing class definitions around retrieved methods, so that class-level context remains available to the agent even though retrieval operates at function granularity. Nodes outside the ancestor set are removed; files with no surviving top-level definitions are deleted. The resulting trimmed repository is

$$\tilde{\mathcal{R}} = \text{Trim}(\mathcal{R}, C_K), \tag{3}$$

which the agent explores in Stage 2.

The retrieval step requires a single forward pass per issue through ϕ and cosine scoring against a precomputed index, adding negligible latency relative to the agent's multi-turn interaction. We analyze the sensitivity to K in §5.2.

3.2 Stage 2. Bounded-Turn Agent Verification

The second stage uses an RL-trained language-model agent to verify and refine the candidate set through terminal-based interaction with the trimmed repository.

MDP formulation. We formulate the agent's interaction as a finite-horizon Markov Decision Process $(\mathcal{S}, \mathcal{A}, P, r, T)$. At each turn t, the state $s_t = (\mathcal{I}, \tilde{\mathcal{R}}, h_{<t})$ comprises the issue text, the trimmed repository, and the interaction history $h_{<t} = (a_0, o_0, \ldots, a_{t-1}, o_{t-1})$ of prior actions and observations. The action space $\mathcal{A}$ consists of two tools executed within the OpenHands environment (Wang et al., 2025a): a `Terminal` tool that runs standard Unix commands (e.g., `find`, `rg` (Gallant, 2016), `cat`) and a `LocalizationFinish` tool that terminates the episode with prediction $y = (F, M, U)$. Following Sutawika et al. (2026), no language-specific static analysis is used; the agent relies solely on text-based exploration of the trimmed repository. The environment returns observation $o_t = P(s_t, a_t)$ (the command output), and the episode ends when the agent calls `LocalizationFinish` or reaches the turn budget T.

Verifiable reward. The agent receives a sparse reward only at episode termination. The prediction

$y = (F, M, U)$ is compared against the ground truth $y^\star = (F^\star, M^\star, U^\star)$ via F1 at each granularity, and the reward is the sum

$$r(y, y^\star) = \mathrm{F1}_{\text{file}}(F, F^\star) + \mathrm{F1}_{\text{mod}}(M, M^\star) + \mathrm{F1}_{\text{func}}(U, U^\star), \quad (4)$$

where $\mathrm{F1}(A, B) = \frac{2|A \cap B|}{|A|+|B|}$ is the set-level F1 score. Each term is independently verifiable against the gold patch, requiring no learned critic or external judge. The reward lies in $[0, 3]$, with 3 indicating perfect localization at all granularities.

3.3 Training Algorithm

We train with Group Sequence Policy Optimization (GSPO) (Zheng et al., 2025), implemented in SkyRL (Cao et al., 2025; Griggs et al., 2025) with asynchronous rollout generation (Fu et al., 2026) and vLLM (Kwon et al., 2023) for inference.

Policy objective. For each training instance $(\mathcal{I}_i, \mathcal{R}_i)$, we sample G rollout trajectories $\{\tau_i^{(1)}, \ldots, \tau_i^{(G)}\}$ from the current policy π_θ. Each trajectory $\tau_i^{(j)}$ is a sequence of state-action-observation tuples yielding a terminal reward $r_i^{(j)}$ via Eq. 4. Following DR.GRPO (Liu et al., 2025), we compute advantages by mean-only normalization within each group,

$$\hat{A}_i^{(j)} = r_i^{(j)} - \bar{r}_i, \quad \bar{r}_i = \tfrac{1}{G}\textstyle\sum_{k=1}^{G} r_i^{(k)}, \quad (5)$$

without dividing by the standard deviation or applying KL regularization. The policy is updated via the clipped surrogate objective applied at the token level across the full trajectory,

$$\mathcal{L}(\theta) = -\mathbb{E}_{i,j}\left[\sum_{t=1}^{T_j} \min\left(\rho_t^{(j)}\hat{A}_i^{(j)},\ \mathrm{clip}(\rho_t^{(j)}, 1-\epsilon, 1+\epsilon)\hat{A}_i^{(j)}\right)\right] \quad (6)$$

where ϵ is the clipping threshold and $\rho_t^{(j)}$ is the importance ratio,

$$\rho_t^{(j)} = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}.$$

Inference-time flexibility. Because pre-filtering is applied only at inference time, the agent is trained on full repositories and the retriever, retrieval budget K, and trimming granularity can all be changed without retraining the policy. This separation is possible because the trimmed repository $\tilde{\mathcal{R}}$ is a strict subset of the original: no new content is introduced, the Unix tools in the action space operate identically on the reduced file set, and the AST rewriting preserves enclosing class scopes so the agent observes the same structural context it would encounter in the full repository.

4 Experimental Setup

4.1 Training Setup

We train on a filtered subset of SWE-Smith (Yang et al., 2026) (39K instances across 128 repositories, no overlap with evaluation benchmarks), using the data curation pipeline of Sutawika et al. (2026) with ground-truth extraction from LocAgent (Chen et al., 2025).

We initialize from `Qwen3-4B-Instruct-2507` (Yang et al., 2025) and train with GSPO (§3.3) for 200 steps on 1.6K instances, sampling 8 rollouts per instance with temperature 1.0. Each episode allows up to 6 turns within a 40K-token context window, scored by the reward in Eq. 4. We use the AdamW optimizer (Loshchilov and Hutter, 2017) with a constant learning rate of $1\mathrm{e}{-6}$ and a batch size of 8. Training runs on 2 NVIDIA B200 GPUs, one for vLLM inference and one for FSDP2 policy optimization.

The agent is trained on the full (untrimmed) repository; pre-filtering is applied only at inference time (§3).

4.2 Evaluation Setup

We evaluate on SWE-Bench Verified (Chowdhury et al., 2024), the Python subset of SWE-Bench Pro (Deng et al., 2025), and SWE-Bench Lite (Jimenez et al., 2024). Ground-truth locations are extracted using the procedure described in §4.1.

For the retrieval stage, we encode all function-level entities at each instance's base commit using Nomic Embed Code (Nussbaum et al., 2024) with bfloat16 precision and a 2,048-token maximum sequence length. We set K based on the retrieval recall analysis in §3. During agent evaluation, we allow up to 15 turns with a context length of 132K tokens. We use the Qwen3 recommended decoding hyperparameters with thinking disabled: temperature 0.7, top-p 0.8, and top-k 20. On the final turn, the system prompt appends a submission reminder instructing the agent to call `LocalizationFinish` immediately. This reminder is applied uniformly to all models (ours, baselines, and closed-source LLMs) to ensure a fair comparison.

Evaluation metrics. Our primary metric is instance-wise average F1 between predicted and ground-truth locations at three granularities (file, module, and function). We also report precision and recall for detailed comparisons.

Baselines. Table 1 compares PILOT against three groups. (1) **Retrieval-only:** Nomic Embed Code (Nussbaum et al., 2024) (137M) and BM25, treating the top-k retrieved functions as the final prediction without agentic verification. (2) **Open-source agentic systems:** RepoNavigator (Zhang et al., 2025) (Qwen2.5-32B, RL-trained with language-server tools), CodeScout (Sutawika et al., 2026) (Qwen3-4B-Instruct, RL with minimal scaffold), and the base `Qwen3-4B-Instruct` without RL on the OpenHands-Bash scaffold. (3) **Closed-source LLMs:** GPT-5 and Claude-Sonnet-4.5 on the same OpenHands-Bash scaffold. The full comparison including Agentless (Xia et al., 2024), LocAgent (Chen et al., 2025), OrcaLoca (Yu et al., 2025), and CoSiL (Jiang et al., 2025) is in Tables 5, 6, and 7 (Appendix). All external baseline numbers are taken from the corresponding published work.

5 Results

5.1 Main Results

On SWE-Bench Verified, PILOT achieves 47.56 function-level F1 with a 4B model, exceeding CodeScout-4B (36.78) by +10.8 points and RepoNavigator-32B (34.09, a model 8× larger) by +13.5 points. Against closed-source systems, PILOT narrows the gap to GPT-5 (54.79) and Claude-Sonnet-4.5 (61.78), both using frontier models orders of magnitude larger.

On SWE-Bench Pro, the most challenging split, PILOT achieves 33.40 function F1, exceeding RepoNavigator-32B (20.72) by +12.7 points and CodeScout-4B (29.03) by +4.4 points.

On SWE-Bench Lite, PILOT achieves 45.84 function F1, surpassing CodeScout-4B (39.87) by +6.0 points and the 18× larger Qwen3-32B (Thinking) (23.76) by +22.1 points.

The improvements are consistent across all three granularity levels: PILOT improves not only function-level F1 but also file and module F1 on every split (Table 4). Examining precision and recall separately, the gains are driven primarily by higher precision, indicating that the agent makes fewer spurious predictions when operating on a concentrated search space.

The gains stem from the two-stage decomposition: retrieval reduces the search space by 32 to 69× (from thousands of functions to K), allowing the RL agent to focus on verification rather than discovery. Comparing against the same model without pre-filtering isolates this effect, showing consistent function F1 improvements of +2.5 to +3.6 pp across splits (Table 4).

5.2 Sensitivity to Retrieval Budget K

Figure 2 plots function-level F1 against the retrieval budget $K \in \{1, 5, 10, 50, 100, 150\}$. Performance rises steeply from $K{=}1$ to $K{=}5$ and plateaus from $K{=}10$ onward, indicating that even a small candidate set suffices for the agent to localize effectively. All K values improve over the full-repository baseline, and the gains are consistent across splits (Appendix C). Based on this analysis, Table 1 reports results with the best K per split, while the ablation in Table 4 fixes $K{=}150$ for controlled comparison.

6 Analysis

6.1 Inference Efficiency

Table 2 compares per-instance inference time between PILOT and the agent without pre-filtering, aggregated over 3 seeds. Pre-filtering reduces median inference time by 8 to 25% across all three splits, with the largest reduction on Verified (−24.7%) where context reduction is most aggressive. The retrieval stage itself adds negligible overhead (a single embedding forward pass and cosine scoring against a precomputed index), so the net effect is a reduction in wall-clock time alongside the accuracy improvements reported above.

6.2 Downstream Issue Resolution

To verify that improved localization translates into practical gains, we feed PILOT's predicted locations into a Qwen3-4B-Instruct resolver on SWE-Bench Verified (Table 3). Concretely, PILOT's predictions are formatted as structured file, class, and function blocks and prepended to the resolver's prompt as relevant context. The resolver is instructed to prioritize the provided locations but not to treat them as exhaustive. Augmenting the resolver prompt with PILOT localizations raises the resolution rate from 9.0% to 15.6%, recovering most of the oracle upper bound (17.8%). The remaining gap to oracle (2.2 pp) suggests that localization quality is close to the ceiling for this resolver configuration. This +6.6 pp gain demon-

Method	LLM	File-level			Module-level			Function-level		
		F1	P	R	F1	P	R	F1	P	R
SWE-Bench Verified										
Retrieval										
Nomic Embed Code (137M) @1	–	58.34	62.41	56.72	42.24	47.34	40.37	32.63	37.84	31.09
BM25 @1	–	33.14	35.87	32.08	24.17	27.59	23.00	20.64	24.32	19.55
Closed-source LLMs										
OpenHands-Bash	GPT-5	78.18	79.25	80.80	61.17	62.23	63.35	54.79	56.80	56.53
OpenHands-Bash	Claude-Sonnet-4.5	82.01	84.50	82.86	67.19	70.11	67.47	61.78	65.42	61.99
Open-source LLMs										
RepoNavigator	Qwen2.5-32B	67.75	70.76	67.29	–	–	–	34.09	37.19	33.71
OpenHands-Bash	Qwen3-4B-Instruct	49.73	49.69	53.34	19.32	19.86	20.15	13.27	13.17	13.74
CodeScout	Qwen3-4B-Instruct	68.52	71.53	67.74	45.97	49.70	44.97	36.78	40.71	35.72
PILOT	**Qwen3-4B-Instruct**	**70.07**	**73.32**	**69.44**	**56.76**	**60.98**	**55.77**	**47.56**	**52.44**	**46.35**
SWE-Bench Pro										
Retrieval										
Nomic Embed Code (137M) @1	–	52.55	75.10	45.52	34.33	64.57	27.40	25.75	57.71	19.73
BM25 @1	–	31.06	46.30	26.42	20.20	37.80	15.96	14.82	32.02	11.50
Closed-source LLMs										
OpenHands-Bash	GPT-5	61.18	69.10	62.06	42.20	52.86	39.63	35.86	48.91	32.65
OpenHands-Bash	Claude-Sonnet-4.5	64.75	75.39	64.49	48.54	61.75	45.39	42.26	58.14	38.06
Open-source LLMs										
RepoNavigator	Qwen2.5-32B	57.57	68.69	53.49	–	–	–	20.72	29.44	18.13
OpenHands-Bash	Qwen3-4B-Instruct	36.96	44.42	35.59	11.78	17.46	10.19	8.12	12.16	7.01
CodeScout	Qwen3-4B-Instruct	51.77	68.98	46.16	36.97	56.05	31.13	29.03	48.65	23.73
PILOT	**Qwen3-4B-Instruct**	**57.63**	**76.61**	**51.66**	**42.07**	**65.78**	**35.49**	**33.40**	**55.81**	**27.74**
SWE-Bench Lite										
Retrieval										
Nomic Embed Code (137M) @1	–	50.67	50.67	50.67	38.61	39.06	38.38	28.71	30.29	27.98
BM25 @1	–	36.00	36.00	36.00	24.35	24.92	24.07	17.18	18.25	16.76
Closed-source LLMs										
OpenHands-Bash	GPT-5	77.77	75.73	82.48	67.86	66.09	72.63	61.12	59.43	67.21
OpenHands-Bash	Claude-Sonnet-4.5	81.87	80.17	85.40	69.62	68.87	72.87	61.11	61.72	63.72
Open-source LLMs										
OpenHands-Bash	Qwen3-4B-Instruct	47.41	43.70	55.47	14.50	13.90	16.24	8.72	8.67	9.61
OpenHands-Bash	Qwen3-32B (Thinking)	58.98	54.26	71.53	37.20	34.64	43.98	23.76	23.11	26.89
CodeScout	Qwen3-4B-Instruct	67.03	66.61	67.88	53.10	53.47	53.65	39.87	41.59	39.96
PILOT	**Qwen3-4B-Instruct**	**71.77**	**71.36**	**72.64**	**59.38**	**59.96**	**59.79**	**45.84**	**47.80**	**45.76**

Table 1: Results across SWE-Bench Verified, Pro, and Lite. F1, precision (P), and recall (R) are shown at file, module, and function granularity. Retrieval rows treat the top-k retrieved functions as the final prediction without agentic verification. PILOT adds dense retrieval pre-filtering to the same RL-trained 4B model, selecting the best K per split from $\{1, 5, 10, 50, 100, 150\}$ by seed-averaged function F1 (Figure 2); the full tables with all baselines are in Appendix Tables 5–7. "–" indicates the metric was not reported.

strates that the localization quality measured by F1 carries over to end-to-end issue resolution.

6.3 Ablation Study

Table 4 isolates the contribution of each component by progressively removing one design choice at a time.

Split	w/o Pre-filt.	PILOT	Reduction
Verified	6.00	4.52	−24.7%
Pro	5.85	5.09	−12.9%
Lite	4.97	4.59	−7.7%

Table 2: Median per-instance inference time (seconds) over 3 seeds. Pre-filtering reduces inference time across all splits by constraining the agent's search space.

Retrieval granularity. Replacing function-level retrieval with file-level retrieval ("w/ file-level retrieval") degrades performance: function F1 drops

Localization	Resolution Rate ↑
No localization	9.0%
PILOT	**15.6%**
Oracle localization	17.8%

Table 3: Downstream issue resolution on SWE-Bench Verified with Qwen3-4B-Instruct as the resolver. Augmenting the resolver prompt with PILOT localizations recovers most of the oracle upper bound (+6.6 pp over no localization, within 2.2 pp of oracle).

Method	Verified			Pro			Lite		
	File	Module	Func.	File	Module	Func.	File	Module	Func.
PILOT	**70.96**±0.9	**58.08**±0.8	**47.30**±0.7	**57.97**±0.7	**41.24**±0.8	**32.38**±1.0	**69.44**±2.0	**57.42**±2.1	**43.90**±1.5
w/ file-level retrieval	68.50±0.8	53.87±1.0	44.10±0.9	55.61±0.7	38.83±0.9	30.69±0.8	68.56±1.5	55.57±1.8	42.48±1.3
w/o pre-filtering	69.95±0.3	55.77±0.6	44.83±1.4	55.82±0.3	38.20±1.1	29.21±0.4	66.97±1.3	53.36±1.7	40.28±1.4
w/o RL training	54.36±1.1	22.56±1.3	16.12±1.4	44.21±1.7	16.96±0.5	11.54±0.9	53.72±1.6	21.64±1.7	13.84±0.8
OpenHands-Bash	52.01±1.9	20.83±0.3	15.04±0.3	41.60±1.4	15.52±0.3	9.98±0.5	51.32±1.4	19.00±1.6	11.42±0.7

Table 4: Ablation study (F1 %, 3-seed average ± std). Each row removes or replaces one component of PILOT. "w/ file-level retrieval" retains entire files instead of individual functions. "w/o pre-filtering" runs the RL-trained agent on the full repository. "w/o RL training" applies function-level pre-filtering to the vanilla Qwen3-4B-Instruct model. "OpenHands-Bash" is the vanilla Qwen3-4B-Instruct model with no pre-filtering.

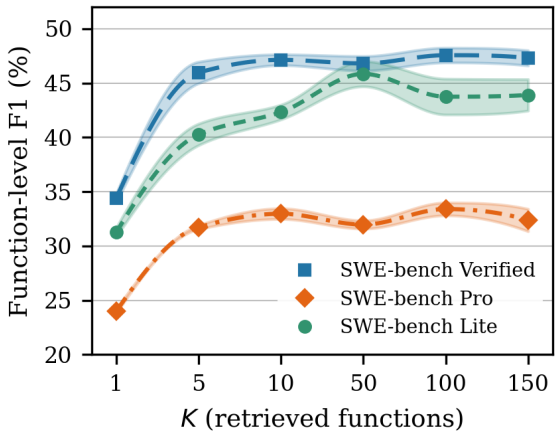

Figure 2: Function-level F1 (%) vs. retrieval budget K across SWE-bench Verified, Pro, and Lite (3-seed average; shaded regions show ±1 std). Performance rises steeply from K=1 to K=5 and plateaus around K=10, indicating that even a small retrieval budget suffices to improve localization.

by −3.2 pp on Verified, −1.7 pp on Pro, and −1.4 pp on Lite. Function-level pre-filtering outperforms file-level despite retaining strictly less content, suggesting that removing intra-file distractors is more beneficial than simply reducing the number of files.

Effect of pre-filtering. Removing pre-filtering entirely ("w/o pre-filtering") reduces function F1 by −2.5 pp on Verified, −3.2 pp on Pro, and −3.6 pp on Lite relative to the full PILOT. The RL-trained agent benefits from the concentrated search space across all splits and granularities.

Effect of RL training. Removing RL training while keeping pre-filtering ("w/o RL training") causes the largest drop: function F1 falls by −31.2 pp on Verified, −20.8 pp on Pro, and −30.1 pp on Lite, confirming that RL-trained verification is the dominant contributor. Comparing the two bottom rows shows that pre-filtering still provides consistent gains even without RL (+1.1 pp on Verified, +1.6 pp on Pro, +2.4 pp on Lite), though the RL-trained agent benefits more from pre-filtering than the vanilla model. Taken together, the ablation indicates that RL training and pre-filtering are complementary: RL provides the dominant capability, but the concentrated search space allows the trained policy to exploit its verification skills more effectively than it can over the full repository.

7 Conclusion

PILOT decouples code localization into retrieval-based discovery and RL-based verification. A single embedding pass reduces the function-level search space by 32 to 69×, and a 4B-parameter agent improves function-level F1 by 2.5 to 3.6 pp across three SWE-Bench splits with 8 to 25% lower median inference time. Ablations show that function-level AST trimming outperforms file-level filtering by 1.4 to 3.2 pp despite retaining strictly less content, and that the RL-trained agent benefits more from pre-filtering than a vanilla model. Future work includes training directly on trimmed repositories, adaptive retrieval budgets, alternative retrievers, and extending beyond Python.

Limitations and Future Work

We evaluate exclusively on Python repositories. Extending to other languages requires language-specific entity extraction and new training corpora. We validate at a single model scale (4B parameters); although PILOT already surpasses models 8× larger, behavior at other scales is untested. On the retrieval side, we use a single dense retriever; hybrid retrieval and alternative code-embedding models remain unexplored, though the modular design admits drop-in replacements without retraining.

Natural next steps include training on trimmed repositories to align the train and inference distributions, scaling to larger policy models, and supporting multilingual repositories.

Ethics Statement

Our work improves code localization, a capability underlying automated issue resolution. Progress in coding agents can reshape the software-engineering profession; we advocate responsible deployment with appropriate human oversight. All data are derived from publicly released benchmarks and open-source GitHub repositories; no private or personally identifying data are introduced.

AI assistance disclosure. AI-based writing assistants were used for editing and proofreading portions of this manuscript. All experimental design decisions, implementations, and result interpretations are the work of the authors.

References

Shiyi Cao, Sumanth Hegde, Dacheng Li, Tyler Griggs, Shu Liu, Eric Tang, Jiayi Pan, Xingyao Wang, Akshay Malik, Graham Neubig, and 1 others. 2025. Skyrl-v0: Train real-world long-horizon agents via reinforcement learning. *arXiv preprint arXiv:2502.02789*.

Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. 2025. Locagent: Graph-guided llm agents for code localization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8697–8727.

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. 2024. Introducing SWE-bench verified.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, and 1 others. 2025. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, and 1 others. 2026. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *Advances in Neural Information Processing Systems*, 38:36256–36282.

Andrew Gallant. 2016. ripgrep. https://github.com/BurntSushi/ripgrep.

Tyler Griggs, Sumanth Hegde, Eric Tang, Shu Liu, Shiyi Cao, Dacheng Li, Charlie Ruan, Philipp Moritz, Kourosh Hakhamaneshi, Richard Liaw, Akshay Malik, Shishir G. Patil, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Evolving SkyRL into a highly-modular RL framework. NovaSky Blog.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. 2025. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. In *NeurIPS 2025 Fourth Workshop on Deep Learning for Code*.

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. *arXiv preprint arXiv:2503.22424*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding r1-zero-like training: A critical perspective. In *Second Conference on Language Modeling*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Michael Luo, Naman Jain, Jaskirat Singh, Sahaj Tan, Aman Patel, Qing Wu, Aria Ariyak, Cheng Cai, Tanay Venkat, Sijun Zhu, Ben Athiwaratkun, Manan Roongta, Charles Zhang, Eugene Lawrence Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. 2025. Deepswe: Training a fully open-sourced, state-of-the-art coding agent by scaling rl. Together AI Blog.

Zexiong Ma, Chao Peng, Qunhong Zeng, Pengfei Gao, Yanzhen Zou, and Bing Xie. 2025. Tool-integrated reinforcement learning for repo deep search. *arXiv preprint arXiv:2508.03012*.

Zach Nussbaum, John X Morris, Brandon Duderstadt, and Andriy Mulyar. 2024. Nomic embed: Training a reproducible long context text embedder. *arXiv preprint arXiv:2402.01613*.

Ben Pan, Carlo Baronio, Albert Tam, Pietro Marsella, Mokshit Jain, Daniel Chiu, Swyx, and Silas Alberti. 2025a. Introducing SWE-grep and SWE-grep-mini: RL for multi-turn, fast context retrieval. Cognition Blog.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2025b. Training software engineering agents and verifiers with swe-gym. In *International Conference on Machine Learning*, pages 47717–47737. PMLR.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389.

Lintang Sutawika, Aditya Bharat Soni, Apurva Gandhi, Taha Yassine, Sanidhya Vijayvargiya, Yuchen Li, Xuhui Zhou, Yilin Zhang, Leander Melroy Maben, Graham Neubig, and 1 others. 2026. Codescout: An effective recipe for reinforcement learning of code search agents. *arXiv preprint arXiv:2603.17829*.

Xingyao Wang, Simon Rosenberg, Juan Michelini, Calvin Smith, Hoang Tran, Engel Nyst, Rohit Malhotra, Xuhui Zhou, Valerie Chen, Robert Brennan, and 1 others. 2025a. The openhands software agent sdk: A composable and extensible foundation for production agents. *arXiv preprint arXiv:2511.03690*.

Zora Zhiruo Wang, Akari Asai, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2025b. Coderagbench: Can retrieval augment code generation? In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3199–3214.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida Wang. 2026. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *Advances in Neural Information Processing Systems*, 38:78500–78525.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

John Yang, Kilian Lieret, Carlos Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. 2026. Swe-smith: Scaling data for software engineering agents. *Advances in Neural Information Processing Systems*, 38.

Zhongming Yu, Hejia Zhang, Yujie Zhao, Hanxian Huang, Matrix Yao, Ke Ding, and Jishen Zhao. 2025. Orcaloca: An llm agent framework for software issue localization. In *International Conference on Machine Learning*, pages 73416–73436. PMLR.

Zhaoxi Zhang, Yitong Duan, Yanzhi Zhang, Yiming Xu, Zhixiang Wang, Kun Liang, Yang Li, Jiahui Liang, Deguo Xia, Jizhou Huang, and 1 others. 2025. One tool is enough: Reinforcement learning for repository-level llm agents. *arXiv preprint arXiv:2512.20957*.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, and 1 others. 2025. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*.

A Training Details

The RL reward curve for the 4B model rises across 200 training steps (Figure 3). We disable entropy loss and mask loss on rollouts that exhaust the maximum number of steps without calling `LocalizationFinish`; both were stabilizing choices in our pilots.

Figure 3: Training reward (sum of file, module, and function F1) over 200 GSPO steps. The reward trends upward throughout training with variance typical of RL on multi-turn tasks.

Software versions. We list all software dependencies for reproducibility: vLLM (Kwon et al., 2023) (v0.11.0), SkyRL (Cao et al., 2025) (v0.2.0), OpenHands (Wang et al., 2025a) (v1.7.1), Nomic Embed Code (`nomic-embed-code-v1`), PyTorch 2.x with FSDP2, and Python 3.11. All experiments use CUDA 12.x on NVIDIA B200 GPUs.

B Full Comparison Table

Tables 5, 6, and 7 present the complete comparison for each split, including all retrieval cutoffs, all open-source baselines, and the controlled w/o pre-filtering rows omitted from the condensed Table 1

in the main text. All "Ours" rows are seed 0; CodeScout numbers are from Sutawika et al. (2026). "–" indicates the metric was not reported.

C Detailed Results

Table 8 reports per-seed F1 at all three granularities for the agent without pre-filtering and PILOT (K=150), along with the per-seed delta. All nine seed-averaged cells are positive.

Method	LLM	File-level			Module-level			Function-level		
		F1	P	R	F1	P	R	F1	P	R
Retrieval										
Nomic Embed Code (137M) @1	–	58.34	62.41	56.72	42.24	47.34	40.37	32.63	37.84	31.09
Nomic Embed Code (137M) @5	–	46.90	37.53	77.00	32.10	22.84	66.72	23.31	15.62	58.69
Nomic Embed Code (137M) @10	–	34.60	24.02	83.84	22.46	14.06	74.64	15.31	9.19	66.16
BM25 @1	–	33.14	35.87	32.08	24.17	27.59	23.00	20.64	24.32	19.55
BM25 @2	–	33.99	31.20	41.67	24.73	22.28	31.62	19.80	17.43	26.64
BM25 @5	–	29.23	21.49	57.24	20.38	14.80	42.86	14.30	9.73	36.01
Closed-source LLMs										
OpenHands-Bash	GPT-5	78.18	79.25	80.80	61.17	62.23	63.35	54.79	56.80	56.53
OpenHands-Bash	Claude-Sonnet-4.5	82.01	84.50	82.86	67.19	70.11	67.47	61.78	65.42	61.99
Open-source LLMs with specialized scaffolds										
Agentless	Qwen2.5-32B	35.38	25.60	78.93	–	–	–	27.33	24.07	40.97
LocAgent	Qwen2.5-32B	44.18	34.18	79.39	–	–	–	21.48	16.29	46.79
OrcaLoca	Qwen2.5-32B	58.11	59.51	59.57	–	–	–	28.72	25.59	39.14
CoSiL	Qwen2.5-32B	30.77	19.34	83.50	–	–	–	22.11	14.85	55.38
RepoNavigator	Qwen2.5-7B	51.63	53.83	50.62	–	–	–	27.49	30.34	26.69
RepoNavigator	Qwen2.5-14B	58.90	58.97	61.60	–	–	–	29.23	30.08	31.02
RepoNavigator	Qwen2.5-32B	67.75	70.76	67.29	–	–	–	34.09	37.19	33.71
Open-source LLMs with OpenHands-Bash scaffold										
OpenHands-Bash	Qwen3-4B-Instruct	49.73	49.69	53.34	19.32	19.86	20.15	13.27	13.17	13.74
CodeScout	Qwen3-4B-Instruct	68.52	71.53	67.74	45.97	49.70	44.97	36.78	40.71	35.72
Ours (w/o pre-filt.)	Qwen3-4B-Instruct	70.05	73.6	69.1	55.42	59.7	54.1	43.80	48.4	42.7
PILOT	**Qwen3-4B-Instruct**	**70.75**	**74.2**	**69.9**	**57.19**	**61.2**	**56.2**	**48.25**	**53.0**	**47.3**

Table 5: Full results on **SWE-Bench Verified** (K=150, seed 0).

Method	LLM	File-level			Module-level			Function-level		
		F1	P	R	F1	P	R	F1	P	R
Retrieval										
Nomic Embed Code (137M) @1	–	52.55	75.10	45.52	34.33	64.57	27.40	25.75	57.71	19.73
Nomic Embed Code (137M) @5	–	50.38	49.27	66.58	36.91	36.79	50.30	30.67	30.75	43.86
Nomic Embed Code (137M) @10	–	40.73	33.40	72.98	30.84	24.45	61.77	26.25	20.91	56.07
BM25 @1	–	31.06	46.30	26.42	20.20	37.80	15.96	14.82	32.02	11.50
BM25 @2	–	34.65	39.69	37.71	22.03	30.12	21.87	16.71	25.10	16.13
BM25 @5	–	30.61	26.67	49.17	20.77	19.76	31.31	16.44	16.44	24.18
Closed-source LLMs										
OpenHands-Bash	GPT-5	61.18	69.10	62.06	42.20	52.86	39.63	35.86	48.91	32.65
OpenHands-Bash	Claude-Sonnet-4.5	64.75	75.39	64.49	48.54	61.75	45.39	42.26	58.14	38.06
Open-source LLMs with specialized scaffolds										
Agentless	Qwen2.5-32B	20.07	13.89	43.07	–	–	–	7.98	7.31	11.08
CoSiL	Qwen2.5-32B	20.95	14.03	48.87	–	–	–	7.67	6.00	14.03
RepoNavigator	Qwen2.5-7B	39.74	48.13	36.36	–	–	–	14.29	21.26	12.33
RepoNavigator	Qwen2.5-14B	49.72	58.64	46.85	–	–	–	18.06	25.25	16.05
RepoNavigator	Qwen2.5-32B	57.57	68.69	53.49	–	–	–	20.72	29.44	18.13
Open-source LLMs with OpenHands-Bash scaffold										
OpenHands-Bash	Qwen3-4B-Instruct	36.96	44.42	35.59	11.78	17.46	10.19	8.12	12.16	7.01
CodeScout	Qwen3-4B-Instruct	51.77	68.98	46.16	36.97	56.05	31.13	29.03	48.65	23.73
Ours (w/o pre-filt.)	Qwen3-4B-Instruct	56.30	76.3	49.8	38.85	63.1	32.0	29.36	52.3	23.8
PILOT	**Qwen3-4B-Instruct**	**58.29**	**78.0**	**52.1**	**40.52**	**64.8**	**33.6**	**31.87**	**55.0**	**26.2**

Table 6: Full results on **SWE-Bench Pro** (K=150, seed 0).

Method	LLM	File-level			Module-level			Function-level		
		F1	P	R	F1	P	R	F1	P	R
Retrieval										
Nomic Embed Code (137M) @1	–	50.67	50.67	50.67	38.61	39.06	38.38	28.71	30.29	27.98
Nomic Embed Code (137M) @5	–	41.91	31.59	73.00	28.00	18.97	60.89	19.28	11.97	53.28
Nomic Embed Code (137M) @10	–	31.23	20.45	81.00	20.08	11.85	73.96	12.77	7.19	62.83
BM25 @1	–	36.00	36.00	36.00	24.35	24.92	24.07	17.18	18.25	16.76
BM25 @2	–	34.22	29.83	43.00	21.83	18.35	28.79	15.26	11.86	22.23
BM25 @5	–	26.21	18.32	53.67	16.41	10.84	38.33	10.89	6.72	30.96
Closed-source LLMs										
OpenHands-Bash	GPT-5	77.77	75.73	82.48	67.86	66.09	72.63	61.12	59.43	67.21
OpenHands-Bash	Claude-Sonnet-4.5	81.87	80.17	85.40	69.62	68.87	72.87	61.11	61.72	63.72
Open-source LLMs with OpenHands-Bash scaffold										
OpenHands-Bash	Qwen3-4B-Instruct	47.41	43.70	55.47	14.50	13.90	16.24	8.72	8.67	9.61
OpenHands-Bash	Qwen3-32B (Thinking)	58.98	54.26	71.53	37.20	34.64	43.98	23.76	23.11	26.89
CodeScout	Qwen3-4B-Instruct	67.03	66.61	67.88	53.10	53.47	53.65	39.87	41.59	39.96
Ours (w/o pre-filt.)	Qwen3-4B-Instruct	67.45	66.9	68.6	54.28	54.7	54.9	40.20	41.8	40.4
PILOT	**Qwen3-4B-Instruct**	**72.00**	**71.8**	**72.3**	**59.42**	**60.1**	**59.7**	**45.57**	**47.6**	**45.4**

Table 7: Full results on **SWE-Bench Lite** (K=150, seed 0).

Split	Seed	File F1			Module F1			Function F1		
		w/o PF	PILOT	Δ	w/o PF	PILOT	Δ	w/o PF	PILOT	Δ
Verified	0	70.05	70.75	+0.7	55.42	57.19	+1.8	43.80	48.25	+4.5
	1	69.59	69.96	+0.4	55.32	57.84	+2.5	43.93	46.58	+2.6
	2	70.20	72.17	+2.0	56.56	59.22	+2.7	46.77	47.07	+0.3
	avg	*69.95*	*70.96*	*+1.0*	*55.77*	*58.08*	*+2.3*	*44.83*	*47.30*	*+2.5*
Pro	0	56.30	58.29	+2.0	38.85	40.52	+1.7	29.36	31.87	+2.5
	1	55.57	58.56	+3.0	39.04	42.36	+3.3	29.61	33.82	+4.2
	2	55.60	57.06	+1.5	36.70	40.84	+4.1	28.66	31.45	+2.8
	avg	*55.82*	*57.97*	*+2.1*	*38.20*	*41.24*	*+3.0*	*29.21*	*32.38*	*+3.2*
Lite	0	67.45	72.00	+4.6	54.28	59.42	+5.1	40.20	45.57	+5.4
	1	65.24	67.11	+1.9	50.95	54.60	+3.6	38.55	42.03	+3.5
	2	68.23	69.22	+1.0	54.84	58.25	+3.4	42.09	44.09	+2.0
	avg	*66.97*	*69.44*	*+2.5*	*53.36*	*57.42*	*+4.1*	*40.28*	*43.90*	*+3.6*

Table 8: Per-seed F1 (%) for the agent without pre-filtering (w/o PF) and PILOT (K=150). The seed-averaged delta is positive across all nine split×granularity cells.