OPEN-SWE-TRACES: Advancing Dual-Mode Multilingual Distillation for Software Engineering Agents

Anonymous ACL submission

Abstract

The path toward autonomous software engineering is currently bottlenecked by a severe deficit of diverse, large-scale trajectory data. We address this by introducing OPEN-SWE-TRACES, an expansive dataset of 207,489 agentic trajectories spanning nine programming languages (Python, Go, TS, JS, Rust, Java, PHP, C, C++). Sourced from 20,000 real-world PRs via OpenHands and SWE-agent harnesses, the dataset utilizes a hybrid-reasoning synthesis: Minimax-M2.5 generates trajectories with explicit "thinking" processes, while Qwen3.5-122B provides high-quality behavioral traces. Filtered for permissive licenses (MIT, Apache, BSD) from SWE-rebench-V2, this data facilitates the training of models capable of long-horizon reasoning. We validate the dataset by fine-tuning the Qwen3-30B-A3B series (Thinking, Coder, and Instruct). The best performing model achieves a resolve rates of 60% on SWE-bench Verified, 46.2% on SWE-bench Multilingual, and 36.8% on SWE-bench Pro. These results establish OPEN-SWE-TRACES as a premier resource for distilling human-level software engineering capabilities into efficient, open-source agentic LLMs.

1 Introduction

The landscape of software engineering (SWE) has been fundamentally reshaped by the emergence of Large Language Model (LLM)-driven agents (Cao et al., 2026; Team et al., 2026; MiniMax, 2026; Liu et al., 2025; Google, 2025; OpenAI, 2025; Anthropic, 2025b). These systems do not merely suggest code snippets; they navigate complex repositories, interpret tool feedback, and autonomously iterate on patches. The industry standard for assessing these agents is repository-level issue resolution, popularized by benchmarks such as SWE-bench (Jimenez et al., 2024; Chowdhury et al., 2024), where success is measured by an agent's ability to resolve real-world bugs verified against a project's internal test suite.

As the field expands toward multilingualism with benchmarks like Multi-SWEbench (Zan et al., 2026) and SWE-PolyBench (Rashid et al., 2025), a critical gap has emerged between evaluation and development. While evaluation datasets have become increasingly diverse, the community lacks the large-scale interaction traces and pre-built environments necessary for robust model development. This resource scarcity changed with the release of SWE-Rebench v2 (Badertdinov et al., 2026), which introduced a massive collection of over 32,000 containerized SWE tasks spanning 20 programming languages. Utilizing this unprecedented scale of executable environments, we bridge the gap between static evaluation and large-scale agent training by releasing OPEN-SWE-TRACES: a comprehensive corpus of 207,489 synthesized trajectories across nine programming languages.

OPEN-SWE-TRACES is uniquely designed to facilitate Dual-Mode Multilingual Distillation, a strategy inspired by recent flagship foundation models such as Qwen3.5 (Qwen Team, 2026a) and Qwen3.6 (Qwen Team, 2026b). These systems have pioneered a unified "switchable" reasoning framework, integrating both a thinking mode for complex, multi-step planning and a non-thinking mode for rapid, execution-oriented responses. To support this, we employ an ensemble of state-of-the-art models—specifically MiniMax-M2.5 (MiniMax, 2026) for reasoning-heavy traces and Qwen3.5-122B (Qwen Team, 2026a) for diverse tool-use—to capture both high-fidelity reasoning traces and direct behavioral logs.

The necessity for this distinction is empirically supported by our trajectory statistics, as shown in table 1. We observe that trajectories incorporating internal "thinking" (Chain-of-Thought) demonstrate a remarkable capacity for trajectory compres-

Metric	OpenHands (w/o thinking)	OpenHands (w/ thinking)	SWE-Agent (w/o thinking)	SWE-Agent (w/ thinking)
Total Assistant Tokens	2,078,918,832	1,106,238,984	1,803,168,804	1,816,518,905
Total Assistant Turns	7,225,661	3,529,736	7,705,056	6,788,458
Avg. Assistant Turns per Traj.	94.08	58.22	130.70	74.82
Avg. Tokens per Assistant Turn	287.7	313.4	234.0	267.6

Table 1: Token and turn statistics for OPEN-SWE-TRACES.

sion. Within the OpenHands framework, thinking-enabled traces reduce the average assistant turns per trajectory from 94.08 to 58.22—a 38% increase in execution efficiency. While these thinking steps increase the average tokens per turn, they effectively prune unproductive trial-and-error loops. OPEN-SWE-TRACES allows models to reserve intensive reasoning for difficult problems while maintaining execution speed for standard operations. To ensure technical integrity, our trajectory generation pipeline incorporates rigorous multi-stage quality filtering and a validation process. This framework uses AST-based auditing to eliminate "git hacking" behaviors, where agents attempt to bypass authentic problem-solving by extracting solutions from repository metadata.

The primary contribution of this work is the release of OPEN-SWE-TRACES as a foundational resource for distilling human-level software engineering capabilities into efficient, open-source agentic models. We validate this by fine-tuning the Qwen3-30B-A3B series; our best performing model achieves state-of-the-art resolve rates: 60% on SWE-bench Verified, 46.2% on SWE-bench Multilingual, and 36.8% on SWE-bench Pro. These results demonstrate that OPEN-SWE-TRACES effectively enables the training of the next generation of dual-mode autonomous agents.

Our contributions are summarized as follows:

- **OPEN-SWE-TRACES**: We introduce the largest multilingual trajectory corpus to date, featuring 207,489 high-fidelity agent traces. The dataset spans diverse agent harnesses and is uniquely structured with both "thinking" and "non-thinking" trajectories to support dual-mode agent development.
- **Performant Dual-Mode Distillation**: We release OPEN-SWE-AGENT, fine-tuned from Qwen3-Coder-30B-A3B using OPEN-SWE-TRACES. Our agent demonstrates the efficacy of dual-mode training by achieving a 60% resolve rate on SWE-bench Verified, while maintaining strong performance on SWE-bench Multilingual (46.2%) and SWE-bench Pro (36.8%) benchmarks. These results establish OPEN-SWE-AGENT as a highly capable open-source baseline for autonomous software engineering.
- **Systematic Analysis**: We provide an extensive empirical study isolating the drivers of agentic performance. Our analysis evaluates the impact of base model selection, data filtering strategies (resolved-only vs. all), and the scaling effects of multilingual vs. Python-only distillation. Finally, we analyze the trade-offs between thinking and non-thinking modalities and demonstrate the model's generalization to unseen execution harnesses.

2 OPEN-SWE-TRACES: From Trajectory Synthesis to Quality Filtering

The construction of OPEN-SWE-TRACES follows a systematic pipeline designed to generate high-fidelity, multi-step trajectories across a diverse, multilingual landscape. As illustrated in fig. 1, this workflow transitions from rigorous source selection to large-scale agentic synthesis, culminating in a multi-stage refinement process to ensure that only optimal technically sound problem-solving paths are retained for distillation.

2.1 Repository Selection and Criteria

We anchor our trajectory generation in SWE-rebench v2 (Badertdinov et al., 2026), a multilingual iteration of SWE-rebench (Badertdinov et al., 2025). To ensure both the technical utility and legal integrity of the corpus, we applied the following strict selection heuristics:

- **Multilingual Scope:** We limited instances to nine languages: Python, Java, C, C++, JavaScript, TypeScript, Rust, Go, and PHP.
- **Permissive Licensing:** To remain compliant with redistribution standards, we enforced a

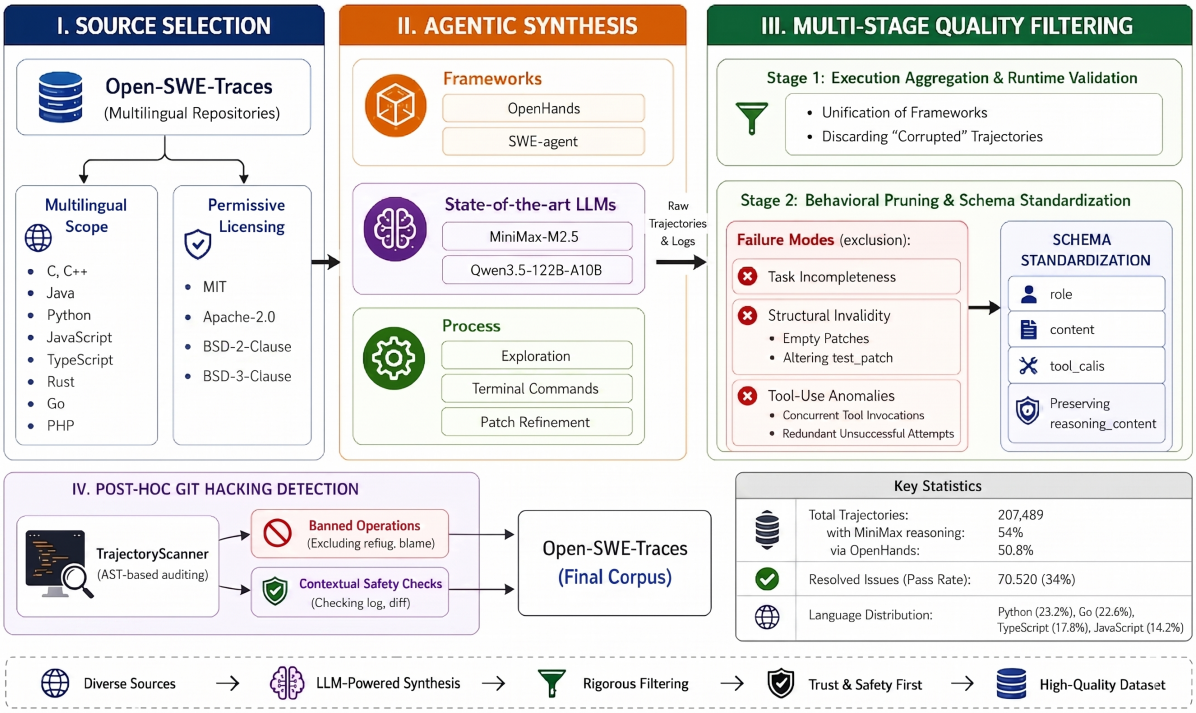

Figure 1: The OPEN-SWE-TRACES Dataset Construction Pipeline.

strict licensing filter. Only repositories distributed under MIT, Apache-2.0, BSD (2-Clause and 3-Clause) licenses were included.

2.2 Multilingual Trajectory Synthesis

To simulate the complexities of real-world software engineering, we employed an ensemble of state-of-the-art LLMs to serve as the cognitive core for autonomous coding agents. Specifically, we integrated MiniMax-M2.5 (MiniMax, 2026) and Qwen3.5-122B-A10B (Qwen Team, 2026a) into the OpenHands (Wang et al., 2025b) and SWE-agent (Yang et al., 2024) frameworks.

Within these environments, the models were tasked with navigating an agent-computer interface to explore unfamiliar codebases, execute terminal commands, and iteratively refine patches for reported issues. By deploying a heterogeneous set of model architectures, we aimed to capture a wider variety of reasoning heuristics and tool-use strategies. This diversity is critical for ensuring that the resulting trajectories represent a robust spectrum of problem-solving approaches rather than the idiosyncratic biases of a single model family.

2.3 Multi-Stage Quality Filtering

To refine raw execution logs into a high-fidelity corpus suitable for model distillation, we implemented a rigorous two-stage filtering and normalization pipeline. This process ensures that only trajectories demonstrating coherent problem-solving and technical correctness are preserved.

Stage 1: Execution Aggregation and Runtime Validation The initial stage focuses on the structural and functional integrity of the generated data. Using a high-performance aggregation utility, we unified the heterogeneous interaction histories produced by the OpenHands and SWE-agent frameworks into a consistent format. During this phase, we prioritize runtime integrity, systematically discarding "corrupted" trajectories where the environment failed to initialize or where agents exhibited catastrophic tool failure (e.g., failing to invoke a bash environment).

Stage 2: Behavioral Pruning and Schema Standardization The second stage of our pipeline serves as a rigorous quality-control gate designed to eliminate trajectories that exhibit suboptimal or "hallucinatory" problem-solving patterns. To ensure the model learns from successful, clean interactions, we systematically prune trajectories based on three primary failure modes:

- Task Incompleteness: We exclude instances where the agent failed to resolve the underlying issue or prematurely reached the maxi-

Language	OpenHands (w/ thinking)	OpenHands (w/o thinking)	SWE-agent (w/ thinking)	SWE-agent (w/o thinking)	Total Traj.
Python	11,790 / 4,608	12,278 / 4,501	13,781 / 4,825	10,330 / 4,086	48,179
Go	12,093 / 4,818	13,324 / 4,827	10,933 / 4,292	10,484 / 4,248	46,834
TypeScript	8,757 / 3,509	9,657 / 3,496	10,600 / 3,714	7,883 / 3,188	36,897
JavaScript	6,854 / 2,786	7,467 / 2,756	8,641 / 3,039	6,398 / 2,585	29,360
Rust	4,412 / 1,975	6,283 / 2,385	6,345 / 2,388	4,695 / 2,024	21,735
Java	3,175 / 1,270	3,463 / 1,288	3,767 / 1,366	2,771 / 1,152	13,176
PHP	2,650 / 1,034	2,747 / 830	2,918 / 1,054	2,008 / 830	10,323
C	159 / 71	184 / 72	197 / 80	143 / 66	683
C++	58 / 27	85 / 33	86 / 33	73 / 32	302
Total	55,488 / 20,098	49,948 / 20,362	57,268 / 20,791	44,785 / 18,211	207,489

Table 2: OPEN-SWE-TRACES statistics across languages. Each cell denotes Trajectories / PRs. Note that PR counts are non-unique across columns and are not intended for horizontal summation.

mum iteration limit without a resolution.

- Structural Invalidity: Trajectories are discarded if they yield empty patches or "cheat" by altering the test suite (`test_patch`) rather than fixing the underlying bug.
- Tool-Use Anomalies: We exclude trajectories characterized by malformed interactions, such as illegal concurrent tool invocations or redundant, unsuccessful attempts.

Following this pruning, the remaining high-fidelity trajectories are standardized into a unified schema. We map framework-specific logs into explicit `role`, `content`, and `tool_calls` fields while crucially preserving the `reasoning_content` from the Minimax-M2.5 model. This ensures the corpus captures the latent chain-of-thought necessary for solving complex software engineering tasks.

2.4 Security and Integrity: Post-hoc Git Hacking Detection

To ensure the integrity of the final corpus, we implemented TrajectoryScanner, a validation framework designed to identify and prune trajectories that exhibit "git hacking." This behavior involves agents attempting to exfiltrate repository metadata or bypass legitimate problem-solving by inspecting internal git histories for solution leaks. By leveraging Abstract Syntax Tree (AST) parsing, the scanner audits every generated Bash command, categorizing them into a risk-based hierarchy. We systematically discard any trajectory containing banned operations that expose sensitive metadata (e.g., `reflog`, `blame`) or restricted commands (e.g., `log`, `diff`) that fail context-specific safety checks. This post-filtering step ensures that the resulting dataset is free from adversarial shortcuts, retaining only those trajectories that demonstrate authentic, safe version control patterns.

2.5 Dataset Statistics

We present the comprehensive statistics for OPEN-SWE-TRACES across its nine supported languages in table 2. The final corpus comprises 207,489 high-fidelity trajectories, of which 54% incorporate internal chain-of-thought reasoning traces generated by the MiniMax-M2.5 model. In terms of framework distribution, 50.8% of the trajectories were synthesized via the OpenHands platform.

The dataset maintains a diverse multilingual balance, with Python (23.2%), Go (22.6%), TypeScript (17.8%), and JavaScript (14.2%) representing the largest shares of the corpus. Finally, we verified the execution correctness of the generated patches through rigorous unit testing; of the total trajectories, 70,520 successfully resolved the underlying issues, yielding an overall pass rate of approximately 34%.

3 Experiments

3.1 Experiment Setup

Agent Scaffolding. We employed the multilingual extension of OpenHands (Wang et al., 2025b) and SWE-agent (Yang et al., 2024), known as MOpenHands (MOH)[1] and MSWE-agent (MSWEA)[2] proposed by Zan et al. (2026). To adapt for multilingual support and improve reliability, both frameworks updated their prompts and

[1] https://github.com/multi-swe-bench/MopenHands
[2] https://github.com/multi-swe-bench/MSWE-agent

Model	Base Model	Harness	Resolved (%)
Proprietary Models			
Claude Opus 4.5 (Anthropic, 2025)	-	-	80.9
GPT-5.2 (xhigh) (OpenAI, 2025)	-	-	80.0
Gemini 3 Pro (Google, 2025)	-	-	76.2
Open Source Foundation Models			
Minimax-M2.5 (MiniMax, 2026)	-	-	80.2
GLM-5 (GLM-5-Team et al., 2026)	-	-	77.8
Kimi-K2.5 (Team et al., 2026)	-	-	76.8
DeepSeek-V3.2 (Liu et al., 2025)	-	-	73.1
Qwen3.5-122B-A10B (Qwen Team, 2026a)	-	-	72.0
Open Source Dense Models of Same Size (32B)			
SWE-Gym-32B (Pan et al., 2025)	Qwen2.5-Coder-32B	OpenHands	20.6
R2E-Gym-32B (Jain et al., 2025)	Qwen2.5-Coder-32B	R2E-Gym	34.4
Skywork-SWE-32B (Zeng et al., 2025b)	Qwen2.5-Coder-32B	OpenHands	38.0
DeepSWE-32B-Preview (Luo et al., 2025)	Qwen3-32B	OpenHands	42.2
SWE-Mirror-LM-32B (Wang et al., 2025a)	Qwen2.5-Coder-32B	MOpenHands	52.2
SWE-Lego (Tao et al., 2026)	Qwen3-32B	OpenHands	52.6
SERA-32B (Shen et al., 2026)	Qwen3-32B	SWE-agent	54.2
daVinci-Dev-32B (Zeng et al., 2026)	Qwen2.5-32B-Base	SWE-agent	56.1
SWE-Master-32B-RL (Song et al., 2026)	Qwen2.5-Coder-32B	R2E-Gym	61.4
SWE-Hero-32B (Ludwig et al., 2026)	Qwen2.5-Coder-32B	OpenHands	62.2
OpenSWE-32B (Fu et al., 2026)	Qwen2.5-32B-Base	SWE-agent	62.4
KAT-Dev-32B (Zhan et al., 2025)	Qwen3-32B	-	62.4
Open Source MoE Models of Same Size (30B-A3B)			
Qwen3-30B-A3B-Inst. (Yang et al., 2025a)	-	OpenHands	22.0
Qwen3-30B-A3B-Think. (Yang et al., 2025a)	-	OpenHands	22.0
Qwen3-Coder-30B-A3B (Yang et al., 2025a)	-	OpenHands	51.6 (50.0*)
GLM-4.7-Flash (Zeng et al., 2025a)	-	-	59.2
Scale-SWE-Agent (Zhao et al., 2026)	Qwen3-30B-A3B-Inst.	OpenHands	64.0 (59.2*)
OPEN-SWE-AGENT-THINKING	Qwen3-30B-A3B-Think.	MSWEA | MOH	55.3 | 51.8
	Qwen3-Coder-30B-A3B	MSWEA | MOH	56.3 | 55.8
OPEN-SWE-AGENT-INSTRUCT	Qwen3-30B-A3B-Inst.	MSWEA | MOH	53.2 | 56.0
	Qwen3-Coder-30B-A3B	MSWEA | MOH	**60.6** | 57.0
OPEN-SWE-AGENT (/think)	Qwen3-Coder-30B-A3B	MSWEA | MOH	58.1 | 58.9
OPEN-SWE-AGENT (/no_think)		MSWEA | MOH	59.8 | **60.0**

Table 3: Performance comparison on the SWE-bench Verified. OPEN-SWE-AGENT performances are reported as x | y, representing Pass@1 using the MSWE-agent (MSWEA) and MOpenHands (MOH) frameworks, respectively. * indicates score calculated by our evaluation pipeline (with MOpenHands agent harness).

added `.gitignore` files to exclude disruptive compiled artifacts. MSWE-agent prioritized stability by truncating long observations and fixing command crashes, while MOpenHands resolved critical bugs—notably a tab-to-space rendering error in `git diff` by utilizing file redirection to ensure patch compatibility in languages like Go.

Agent Distillation. We perform distillation using the Qwen3-30B-A3B (Yang et al., 2025a) as base models, specifically the Thinking, Instruct, and Code variants. To evaluate modality impacts, OPEN-SWE-AGENT-THINKING and OPEN-SWE-AGENT-INSTRUCT are trained on their respective subsets of OPEN-SWE-TRACES, while OPEN-SWE-AGENT leverages the full corpus. The latter supports dual-mode operation via the `/think` and `/no_think` triggers. The training process is configured with a learning rate of 1e-5, a batch size of 32, and a warmup ratio of 0.1, supporting a maximum context length of 131,072.

Evaluation Benchmarks and Metrics. We evaluate performance across three key benchmarks:

Model	Base Model	SWE (M)	SWE (Pro)
Proprietary Models			
Claude Opus 4.5 (Anthropic, 2025)	-	76.2	52.0
GPT-5.2 (xhigh) (OpenAI, 2025)	-	72.0	55.6
Gemini 3 Pro (Google, 2025)	-	65.0	43.3
Open Source Foundation Models			
Minimax-M2.5 (MiniMax, 2026)	-	80.2	55.4
GLM-5 (GLM-5-Team et al., 2026)	-	73.3	55.1
Kimi-K2.5 (Team et al., 2026)	-	73.0	53.8
DeepSeek-V3.2 (Liu et al., 2025)	-	70.2	-
Qwen3.5-122B-A10B (Qwen Team, 2026a)	-	68.3*	53.6*
Open Source MoE Models of Same Size (30B-A3B)			
Qwen3-Coder-30B-A3B (Yang et al., 2025a)	-	33.5*	28.4*
Scale-SWE-Agent (Zhao et al., 2026)	Qwen3-30B-A3B-Inst.	45.2*	-
OPEN-SWE-AGENT-THINKING	Qwen3-30B-A3B-Think.	36.3 | 35.8	31.3 | -
	Qwen3-Coder-30B-A3B	36.3 | 38.7	32.5 | -
OPEN-SWE-AGENT-INSTRUCT	Qwen3-30B-A3B-Inst.	41.4 | 40.0	32.2 | -
	Qwen3-Coder-30B-A3B	**51.1** | **45.5**	34.3 | -
OPEN-SWE-AGENT (/think)	Qwen3-Coder-30B-A3B	41.3 | 39.3	33.4 | -
OPEN-SWE-AGENT (/no_think)		47.2 | 44.8	**36.8** | -

Table 4: Performance comparison on the SWE-Bench Multilingual (M) and Pro benchmarks. * indicates score calculated by our evaluation pipeline. OPEN-SWE-AGENT performance on SWE (M) are reported as x | y, representing Pass@1 using the MSWE-agent and MOpenHands frameworks, respectively.

SWE-bench Verified (Chowdhury et al., 2024), featuring 500 human-validated Python issues; SWE-bench Multilingual (Jimenez et al., 2024), which contains 300 tasks distributed across eight languages (C, C++, Go, Java, JS/TS, PHP, Ruby, and Rust); and SWE-bench Pro (Deng et al., 2025), providing 731 tasks in Python, Go, and JS/TS. While we utilized both agent harnesses for the Verified and Multilingual benchmarks, MOpenHands was excluded from the SWE-bench Pro evaluation due to persistent integration issues; consequently, we report only MSWE-agent results for this benchmark. Our primary metric is the Resolved Rate (%), defined as the percentage of issues successfully fixed. During inference, we increased the context window to 262,144 tokens to better handle large codebases. Inference hyperparameters are detailed in table 5, and all results represent mean performance across three independent runs.

3.2 Experiment Results

Monolingual Evaluation. As shown in table 3, the OPEN-SWE-AGENT-THINKING and OPEN-SWE-AGENT-INSTRUCT variants demonstrate competitive performance on the SWE-bench Verified, , achieving an absolute improvement of over 30.0% compared to their base model counterparts. While OPEN-SWE-AGENT-INSTRUCT lags behind Scale-SWE-Agent by 4 points, this gap is likely attributable to the difference in training volume: Scale-SWE-Agent was trained on approximately 71k Python trajectories across 25k unique pull requests (PRs), whereas OPEN-SWE-AGENT-INSTRUCT utilized only 24k trajectories from 4.5k PRs. Our primary model, OPEN-SWE-AGENT, achieves a 58.9% and 60.0% resolved rate under the `/think` and `/no_think` settings, respectively—an 7.3–8.4% absolute gain over the strong base model's performance of 51.6%.

Multilingual Evaluation. Evaluation results are presented in table 4. We include Scale-SWE-Agent as the primary baseline for SWE-bench Multilingual (SWE-M); however, we could not obtain meaningful scores for this competitor on the more challenging SWE-bench Pro (SWE-Pro) benchmark, as Scale-SWE-Agent failed on most instances. Compared to the monolingual results, we observe even larger absolute improvements for

Hyperparameter	/think	/no_think
Temperature	1.0	0.7
Top-p	0.95	0.8
Top-k	40	20
Presence Penalty	n/a	1.5
Maximum Turn	250	250
Maximum Context Length	256k	256k

Table 5: Inference hyperparameters for OPEN-SWE-AGENT and its variants.

OPEN-SWE-AGENT. On SWE-M, the resolved rate increased from 33.5% to 47.2%, and on SWE-Pro, performance rose from 28.4% to 36.8%. These results underscore the significant value of OPEN-SWE-TRACES in enhancing agentic reasoning across diverse programming languages.

4 Ablation and Analyses

We aim to address the following research questions.

1. **Cross-Harness Generalization**: Does training on trajectories from one agent harness (e.g., OpenHands) generalize effectively to another (e.g., SWE-agent)?
2. **Cross-Lingual Transfer**: To what extent does a Python-only model generalize to multilingual issue resolution tasks?
3. **Data Filtering Impact**: Does the exclusion of trajectories containing unresolved patches impact downstream agentic performance?

The ablation study results, detailed in table 6, provide a critical look at how training configurations influence agentic capabilities across different environments and modalities.

4.1 Cross-Harness Generalization

The results indicate that while generalization between harnesses is possible, there is a consistent performance penalty when switching agent harness. In every tested configuration, shifting from the native training harness to a cross-harness evaluation results in a drop in resolved rates. For example, in the MSWEA $\rightarrow$ MOH configuration under `think` mode (SWE-bench V.), performance drops from 57.7% to 51.9%. The degradation is more pronounced in the Multilingual (SWE-bench M.) benchmark than in the Verified (Python) benchmark. Notably, in the MSWEA $\rightarrow$ MOH (`think`) category for SWE-bench M., the model loses nearly 9 absolute percentage points ($49.9 \rightarrow 41.1$). These findings suggest that training on specific harness trajectories does not generalize perfectly, as agents appear to overfit slightly to the specific interaction patterns, action spaces, or observation formats of their primary training harness.

4.2 Cross-Lingual Transfer

The transition from a Python-only training set to the "full" (multilingual) OPEN-SWE-TRACES demonstrates significant positive transfer, particularly for non-Python tasks. The most striking improvement is seen in the `no-think` mode for SWE-bench M., where moving from Python-only to the full dataset boosts the resolved rate from 36.0% to 47.2% (a $+11.2\%$ absolute gain). Even on the Python-centric SWE-bench V., adding multilingual data does not dilute performance; instead, it provides a slight boost (e.g., $54.9\% \rightarrow 58.1\%$ in `think` mode). Python-only models generalize reasonably well to other languages, but explicit multilingual distillation is essential for peak performance in diverse programming environments, likely due to the model learning more generalized logic-flow and syntax-agnostic debugging strategies.

4.3 Data Filtering Impact

The results suggest that including trajectories with unresolved patches (the full corpus) is generally superior to training exclusively on "resolved-only" successful trajectories. Moving from resolved-only to the full dataset yields improvements across almost all categories. On SWE-bench V. (`think`), we observe a climb from 55.3% to 58.1%. The impact is particularly helpful for multilingual tasks, where the `think` mode score on SWE-bench M. improves from 38.0% to 41.3%. In the `no-think` mode for SWE-bench M., the performance remains stable at 47.2%. These results suggest that "negative" samples and longer trajectories within the full corpus help the model navigate complex states, even without a successful fix. Excluding unresolved trajectories is counterproductive, as the model benefits from exposure to real-world repository interactions regardless of whether a correct patch was ultimately generated in the training sample.

Key Takeaways. Our analysis establishes a clear hierarchy of performance drivers: multilingual data and full trajectory inclusion (unresolved cases included) are the primary catalysts for gains. Conversely, harness consistency remains a technical challenge, careful alignment across evaluation environments for peak performance.

Experiment / Configuration	SWE-bench V.		SWE-bench M.	
	Thinking	No-Thinking	Thinking	No-Thinking
Cross-Harness Generalization				
OH → SWEA	50.7 → 49.0	57.1 → 56.7	38.1 → 32.8	45.6 → 39.2
SWEA → OH	57.7 → 51.9	54.9 → 48.5	49.9 → 41.1	49.8 → 47.6
Cross-Lingual Transfer				
Python-only → Full	54.9 → 58.1	58.6 → 59.8	40.4 → 41.3	36.0 → 47.2
Data Filtering Impact				
Resolved-only → Full	55.3 → 58.1	57.7 → 59.8	38.0 → 41.3	47.2 → 47.2

Table 6: Evaluation results of distillation using different subset of OPEN-SWE-TRACES.

5 Related Works

SWE Benchmarks. The software engineering evaluation landscape has evolved from foundations like SWE-bench (Jimenez et al., 2024) and SWE-bench-Verified (Chowdhury et al., 2024) into a sophisticated ecosystem. Modern benchmarks now probe specialized dimensions including multi-modal integration (Yang et al., 2024), cross-linguistic proficiency (Guo et al., 2025; Rashid et al., 2025; Zan et al., 2026), and long-horizon reasoning (Deng et al., 2025). Contemporary evaluations also scrutinize high-level tasks like full-repository generation (Ding et al., 2025), scientific expertise (Duston et al., 2025), and niche technical competencies (Ma et al., 2025; Shetty et al., 2026), establishing a rigorous standard for autonomous engineering. Notably, BeyondSWE (Chen et al., 2026) extends this scope to cross-repository reasoning and dependency-driven migration, further raising the bar for autonomous engineering.

SWE Datasets. Improving LLM programming proficiency requires high-quality, repository-level data. Data scaling strategies generally follow two paths: synthetic generation (e.g., R2E-Gym (Jain et al., 2025), SWE-smith (Yang et al., 2025b), and SWE-Mirror (Wang et al., 2025a)) and real-world mining. While SWE-Gym (Pan et al., 2025) and SWE-rebench (Badertdinov et al., 2025) established thousands of validated instances, Scale-SWE (Zhao et al., 2026) massively expands this to 100k instances using multi-agent workflows. To optimize training efficiency, SWE-Zero/Hero (Ludwig et al., 2026) utilizes a two-stage distillation pipeline, contributing 300k execution-free and 13k execution-backed trajectories. Similarly, SWE-World (Sun et al., 2026) enables large-scale data utilization by replacing physical environments with learned surrogate models for feedback. Synthesizing these trends, SWE-Lego (Tao et al., 2026) merges authentic PRs with synthetic instances to maximize both precision and training volume.

SWE Models and Agents. Autonomous coding is shifting from external architectures toward model-centric optimization. While early frameworks like SWE-agent (Yang et al., 2024) and OpenHands (Wang et al., 2025b) used environment-based prompting, newer research embeds reasoning into model weights via mid-training (Zeng et al., 2026), SFT on expert trajectories (Wang et al., 2025a; Tao et al., 2026), or reinforcement learning (Luo et al., 2025; Cao et al., 2025). Parallel to these agents, a modular "agentless" paradigm (Xia et al., 2025; Yang et al., 2025c) has emerged, streamlining troubleshooting into distinct stages of localization, repair, and verification (He et al., 2025; Xie et al., 2025). In this landscape, Orchard (Peng et al., 2026) bridges the proprietary performance gap by codifying large-scale agentic trajectories into reusable, open-source training recipes.

6 Conclusion

We introduced OPEN-SWE-TRACES, the most extensive collection of agentic software engineering trajectories to date, featuring 200,000+ traces across nine languages. By addressing the scarcity of large-scale agentic data, this work enables the distillation of complex engineering behaviors into open-source models. Our evaluation shows strong results: a 60% resolve rate on SWE-bench Verified, 46.2% on Multilingual, and 36.8% on the rigorous SWE-bench Pro. These findings suggest that combining explicit thinking modalities with high-quality behavioral traces is a highly effective path for developing specialized autonomous agents.

Limitations

While our distillation framework achieves state-of-the-art results, it is subject to several constraints. First, the performance and behavioral profile of our agents are inherently tethered to the teacher models (Minimax-M2.5 and Qwen3.5-122B-A10B); consequently, any latent biases or systemic errors present in these predecessors are likely inherited by the students. Furthermore, the inherent stochasticity of LLMs, compounded by the volatility of software execution environments, introduces unavoidable variance. Although we mitigated this through triple-run aggregation, infrastructure dependencies and environmental factors can cause minor fluctuations in scores, which may affect precise reproducibility.

Ethics Statement

We are committed to the public release of OPEN-SWE-TRACES and have implemented rigorous protocols to ensure its integrity. All agent trajectories were derived from open-source repositories under permissive licenses (e.g., MIT, Apache 2.0, BSD), with automated filtering pipelines employed to redact Personally Identifiable Information (PII) and sensitive credentials. Furthermore, we utilized Gemini to assist in refining the manuscript's linguistic clarity and to generate the visual data flow illustration (Figure 1). Nonetheless, the authors have meticulously reviewed all suggestions and remain fully responsible for the research findings, technical accuracy, and final content.

References

Anthropic. 2025. Introducing claude opus 4.5. https://www.anthropic.com/news/claude-opus-4-5. Accessed: 2026-03-17.

Anthropic. 2025b. Introducing Claude sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5. Published: 2025-09-29; Accessed: 2025-12-22.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. 2025. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, and Alexander Golubev. 2026. Swe-rebench v2: Language-agnostic swe task collection at scale. *arXiv preprint arXiv:2602.23866*.

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, Zeyao Ma, Kashun Shum, Xuwu Wang, Jinxi Wei, Jiaxi Yang, Jiajun Zhang, Lei Zhang, Zongmeng Zhang, Wenting Zhao, and Fan Zhou. 2026. Qwen3-coder-next technical report. *Preprint*, arXiv:2603.00729.

Shiyi Cao, Dacheng Li, Fangzhou Zhao, Shuo Yuan, Sumanth R. Hegde, Connor Chen, Charlie Ruan, Tyler Griggs, Shu Liu, Eric Tang, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Skyrl-agent: Efficient rl training for multi-turn llm agent. *Preprint*, arXiv:2511.16108.

Guoxin Chen, Fanzhe Meng, Jiale Zhao, Minghao Li, Daixuan Cheng, Huatong Song, Jie Chen, Yuzhi Lin, Hui Chen, Xin Zhao, Ruihua Song, Chang Liu, Cheng Chen, Kai Jia, and Ji-Rong Wen. 2026. Beyondswe: Can current code agent survive beyond single-repo bug fixing. *Preprint*, arXiv:2603.03194.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, and 1 others. 2024. Introducing swe-bench verified. *arXiv preprint arXiv:2407.01489*.

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*.

Jingzhe Ding, Shengda Long, Changxin Pu, Huan Zhou, Hongwan Gao, Xiang Gao, Chao He, Yue Hou, Fei Hu, Zhaojian Li, and 1 others. 2025. Nl2repo-bench: Towards long-horizon repository generation evaluation of coding agents. *arXiv preprint arXiv:2512.12730*.

Titouan Duston, Shuo Xin, Yang Sun, Daoguang Zan, Aoyan Li, Shulin Xin, Kai Shen, Yixiao Chen, Qiming Sun, Ge Zhang, and 1 others. 2025. Ainstein-bench: Benchmarking coding agents on scientific repositories. *arXiv preprint arXiv:2512.21373*.

Dayuan Fu, Shenyu Wu, Yunze Wu, Zerui Peng, Yaxing Huang, Jie Sun, Ji Zeng, Mohan Jiang, Lin Zhang, Yukun Li, Jiarui Hu, Liming Liu, Jinlong Hou, and Pengfei Liu. 2026. davinci-env: Open swe environment synthesis at scale. *Preprint*, arXiv:2603.13023.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, and 168 others. 2026. Glm-5: from vibe coding to agentic engineering. *Preprint*, arXiv:2602.15763.

Google. 2025. Gemini 3 pro. Accessed: 2026-05-21.

Lianghong Guo, Wei Tao, Runhan Jiang, Yanlin Wang, Jiachi Chen, Xilin Liu, Yuchi Ma, Mingzhi Mao, Hongyu Zhang, and Zibin Zheng. 2025. Omnigirl: A multilingual and multimodal benchmark for github issue resolution. *Proceedings of the ACM on Software Engineering*, 2(ISSTA):24–46.

Zhenyu He, Qingping Yang, Wei Sheng, Xiaojian Zhong, Kechi Zhang, Chenxin An, Wenlei Shi, Tianle Cai, Di He, Jiaze Chen, and Jingjing Xu. 2025. SWE-Swiss: A multi-task fine-tuning and RL recipe for high-performance issue resolution. Notion Blog / GitHub Repository. Accessed: 2026-03-17.

Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. 2025. R2e-gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents. In *Second Conference on Language Modeling*.

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*.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, and 244 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

Nikolai Ludwig, Wasi Uddin Ahmad, Somshubra Majumdar, and Boris Ginsburg. 2026. From swe-zero to swe-hero: Execution-free to execution-based fine-tuning for software engineering agents. *arXiv preprint arXiv:2604.01496*.

M. Luo, N. Jain, J. Singh, S. Tan, A. Patel, Q. Wu, A. Ariyak, C. Cai, T. Venkat, S. Zhu, B. Athiwaratkun, M. Roongta, C. Zhang, L. E. Li, R. A. Popa, K. Sen, and I. Stoica. 2025. DeepSWE: Training a fully open-sourced, state-of-the-art coding agent by scaling RL. https://www.together.ai/blog/deepswe. Together AI Blog post. Accessed: 2025-12-22.

Yingwei Ma, Rongyu Cao, Yongchang Cao, Yue Zhang, Jue Chen, Yibo Liu, Yuchen Liu, Binhua Li, Fei Huang, and Yongbin Li. 2025. Swe-gpt: A process-centric language model for automated software improvement. *Proceedings of the ACM on Software Engineering*, 2(ISSTA):2362–2383.

MiniMax. 2026. Minimax m2.5: Built for real-world productivity. https://www.minimax.io/news/minimax-m25. Accessed: 2026-03-17.

OpenAI. 2025. Introducing GPT-5.2. https://openai.com/index/introducing-gpt-5-2/.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2025. Training software engineering agents and verifiers with SWE-gym. In *Forty-second International Conference on Machine Learning*.

Baolin Peng, Wenlin Yao, Qianhui Wu, Hao Cheng, Xiao Yu, Rui Yang, Tao Ge, Alessandrio Sordoni, Xingdi Yuan, Yelong Shen, and 1 others. 2026. Orchard: An open-source agentic modeling framework. *arXiv preprint arXiv:2605.15040*.

Qwen Team. 2026a. Qwen3.5: Towards native multimodal agents.

Qwen Team. 2026b. Qwen3.6-27B: Flagship-level coding in a 27B dense model.

Muhammad Shihab Rashid, Christian Bock, Yuan Zhuang, Alexander Buchholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, and 1 others. 2025. Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents. *arXiv preprint arXiv:2504.08703*.

Ethan Shen, Danny Tormoen, Saurabh Shah, Ali Farhadi, and Tim Dettmers. 2026. Sera: Soft-verified efficient repository agents. *Preprint*, arXiv:2601.20789.

Manish Shetty, Naman Jain, Jinjian Liu, Vijay Kethanaboyina, Koushik Sen, and Ion Stoica. 2026. Gso: Challenging software optimization tasks for evaluating swe-agents. *Advances in Neural Information Processing Systems*, 38.

Huatong Song, Lisheng Huang, Shuang Sun, Jinhao Jiang, Ran Le, Daixuan Cheng, Guoxin Chen, Yiwen Hu, Zongchao Chen, Yiming Jia, Wayne Xin Zhao, Yang Song, Tao Zhang, and Ji-Rong Wen. 2026. Swe-master: Unleashing the potential of software engineering agents via post-training. *Preprint*, arXiv:2602.03411.

Shuang Sun, Huatong Song, Lisheng Huang, Jinhao Jiang, Ran Le, Zhihao Lv, Zongchao Chen, Yiwen Hu, Wenyang Luo, Wayne Xin Zhao, Yang Song, Hongteng Xu, Tao Zhang, and Ji-Rong Wen. 2026. Swe-world: Building software engineering agents in docker-free environments. *Preprint*, arXiv:2602.03419.

Chaofan Tao, Jierun Chen, Yuxin Jiang, Kaiqi Kou, Shaowei Wang, Ruoyu Wang, Xiaohui Li, Sidi Yang, Yiming Du, Jianbo Dai, Zhiming Mao, Xinyu Wang, Lifeng Shang, and Haoli Bai. 2026. Swe-lego: Pushing the limits of supervised fine-tuning for software issue resolving. *Preprint*, arXiv:2601.01426.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, S. H. Cai, Yuan Cao, Y. Charles, H. S. Che, Cheng

Chen, Guanduo Chen, Huarong Chen, Jia Chen, Jiahao Chen, Jianlong Chen, Jun Chen, Kefan Chen, Liang Chen, Ruijue Chen, Xinhao Chen, and 307 others. 2026. Kimi k2.5: Visual agentic intelligence. *Preprint*, arXiv:2602.02276.

Haoran Wang, Zhenyu Hou, Yao Wei, Jie Tang, and Yuxiao Dong. 2025a. SWE-dev: Building software engineering agents with training and inference scaling. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 3742–3761, Vienna, Austria. Association for Computational Linguistics.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2025b. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2025. Demystifying llm-based software engineering agents. *Proc. ACM Softw. Eng.*, 2(FSE).

Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. 2025. SWE-fixer: Training open-source LLMs for effective and efficient GitHub issue resolution. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 1123–1139, Vienna, Austria. Association for Computational Linguistics.

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

John Yang, Carlos E Jimenez, Alex L Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R Narasimhan, and 1 others. 2024. Swe-bench multimodal: Do ai systems generalize to visual software domains? *arXiv preprint arXiv:2410.03859*.

John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. 2025b. SWE-smith: Scaling data for software engineering agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Zonghan Yang, Shengjie Wang, Kelin Fu, Wenyang He, Weimin Xiong, Yibo Liu, Yibo Miao, Bofei Gao, Yejie Wang, Yingwei Ma, Yanhao Li, Yue Liu, Zhenxing Hu, Kaitai Zhang, Shuyi Wang, Huarong Chen, Flood Sung, Yang Liu, Yang Gao, and 2 others. 2025c. Kimi-dev: Agentless training as skill prior for swe-agents. *Preprint*, arXiv:2509.23045.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Li Aoyan, Lu Chen, Xiaojian Zhong, and 1 others. 2026. Multi-swe-bench: A multilingual benchmark for issue resolving. *Advances in Neural Information Processing Systems*, 38.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, and 151 others. 2025a. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *Preprint*, arXiv:2508.06471.

Ji Zeng, Dayuan Fu, Tiantian Mi, Yumin Zhuang, Yaxing Huang, Xuefeng Li, Lyumanshan Ye, Muhang Xie, Qishuo Hua, Zhen Huang, Mohan Jiang, Hanning Wang, Jifan Lin, Yang Xiao, Jie Sun, Yunze Wu, and Pengfei Liu. 2026. davinci-dev: Agent-native mid-training for software engineering. *Preprint*, arXiv:2601.18418.

Liang Zeng, Yongcong Li, Yuzhen Xiao, Changshi Li, Chris Yuhao Liu, Rui Yan, Tianwen Wei, Jujie He, Xuchen Song, Yang Liu, and Yahui Zhou. 2025b. Skywork-swe: Unveiling data scaling laws for software engineering in llms. *Preprint*, arXiv:2506.19290.

Zizheng Zhan, Ken Deng, Jinghui Wang, Xiaojiang Zhang, Huaixi Tang, Minglei Zhang, Zhiyi Lai, Haoyang Huang, Wen Xiang, Kun Wu, and 1 others. 2025. Kat-coder technical report. *arXiv preprint arXiv:2510.18779*.

Jiale Zhao, Guoxin Chen, Fanzhe Meng, Minghao Li, Jie Chen, Hui Xu, Yongshuai Sun, Wayne Xin Zhao, Ruihua Song, Yuan Zhang, and 1 others. 2026. Immersion in the github universe: Scaling coding agents to mastery. *arXiv preprint arXiv:2602.09892*.

Technical Appendices

A Implementation Details

The SFT hyperparameters are detailed in Table 7.

Hyperparameter	Value
Learning Rate	1e-5
Min. Learning Rate	1e-8
Base model	Qwen3-30B-A3B
Batch size	32
Maximum Context Length	131,072
Warmup ratio	0.1
Weight decay	0.01
LR scheduler type	Cosine
Epoch	3

Table 7: Key hyperparameters for distillation.

A.1 Git-hack Detection

The trajectory scanner is provided in Fig. 2.

```python
import json, re, bashlex

class BashParser:
    _HDR = (re.compile(r" << ?'([A-Z]+)'"), r" << \1")

    @classmethod
    def get_commands(cls, cmd):
        try: trees = bashlex.parse(cmd.strip())
        except:
            try: trees = bashlex.parse(cls._HDR[0].sub(cls._HDR[1], cmd))
            except: return
        for tree in (trees or []):
            stack = [tree]
            while stack:
                curr = stack.pop()
                if curr.kind == 'command': yield curr
                if hasattr(curr, 'parts'): stack.extend(reversed(curr.parts))

class GitPolicy:
    BANNED = {"blame", "shortlog", "rev-list", "reflog"}
    INSPECT = {"log", "show", "checkout", "diff"}
    SAFE = [re.compile(r"^HEAD([~^]([0-9]+)?)?$"), re.compile(r"^\.$"),
            re.compile(r"^.*\.[a-z]{1,4}$"), re.compile(r"^.*/$"), re.compile(r"^\.gitignore$")]

    def __init__(self, commit=None):
        self.pfx = commit[:6] if commit and len(commit) >= 6 else None

    def is_safe(self, t):
        if not t or (self.pfx and t.startswith(self.pfx)): return True
        return any(p.fullmatch(t) for p in self.SAFE)

class GitValidator:
    def __init__(self, policy):
        self.policy = policy

    def is_hacking(self, node):
        words = [p.word for p in node.parts if p.kind == 'word']
        if len(words) < 2 or words[0] != "git": return False
        idx = 3 if words[1] == "-C" and len(words) > 3 else 1
        if idx >= len(words): return False
        cmd = words[idx]
        if cmd in self.policy.BANNED: return True
        if cmd in self.policy.INSPECT:
            for w in [w for w in words[idx+1:] if not w.startswith("-")]:
                if not all(self.policy.is_safe(t) for t in re.split(r'[:]|(?:\.\.)', w)):
                    return True
        return False

class TrajectoryScanner:
    def __init__(self, commit=None):
        self.pol, self.git_re = GitPolicy(commit), re.compile(r"\bgit\b")
        self.val = GitValidator(self.pol)

    def scan(self, msgs):
        for tool in [t for m in msgs if m.get("role")=="assistant" for t in (m.get("tool_calls") or
  [])]:
            if tool.get("function", {}).get("name") in {"execute_bash", "bash"}:
                try:
                    c = json.loads(tool["function"]["arguments"]).get("command", "")
                    if self.check_cmd(c): return True
                except: continue
        return False

    def check_cmd(self, c):
        if not self.git_re.search(c): return False
        nodes = list(BashParser.get_commands(c))
        if not nodes: return any(f"git {b}" in c for b in self.pol.BANNED)
        return any(self.val.is_hacking(n) for n in nodes)
```

Figure 2: Trajectory scanner to check if LLM agents attempt git hacking while solving software issues.