XREPOSKILL: LEARNING TRANSFERABLE SKILLS FOR SOFTWARE ENGINEERING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Software engineering agents increasingly use reusable skills distilled from prior experience to resolve repository-level issues, yet such skills often fail to transfer across repositories. A central challenge is that a behavior appearing in a successful trajectory is not necessarily responsible for the successful outcome: it may be genuinely useful, merely incidental, or simply a recurring habit of the model. We introduce XRepoSkill, a trajectory-based approach for learning transferable skills. We represent a skill as a collection of rules, each specifying what action to take and when to take it during issue resolution. XRepoSkill first contrasts successful and failed trajectories of the same agent on the same issue and derives candidate rules from where their execution paths diverge. Each rule is paired with an executable predicate that enables its prescribed behavior to be evaluated systematically on other trajectories. A rule is verified based on its association with successful issue resolution and retained only when its prescribed behavior recurs across multiple repositories; repository-specific variants of the same behavior are then consolidated into transferable rules. For a new issue, XRepoSkill selects relevant rules to guide the agent. We learn skills from publicly released trajectories on the official SWE-bench Verified leaderboard and evaluate them on SWE-bench Pro and DeepSWE using three backbone LLMs from different vendors; none of the evaluation repositories appears in the skill-learning trajectory pool. Against three recent skill learning methods, XRepoSkill achieves the highest issue resolution rate in all six benchmark–LLM combinations. In particular, on the challenging long-horizon DeepSWE benchmark, XRepoSkill improves issue resolution by 10.3 percentage points over the same agent without learned skills and by 5.0 points over the strongest skill-learning baseline.

1 INTRODUCTION

Software engineering agents can resolve software issues by exploring a repository, modifying its code, running tests, and producing a patch (Jimenez et al., 2024; Yang et al., 2024). One increasingly popular way to equip such agents with reusable expertise, without retraining the backbone LLM, is through *skills* (Anthropic, 2026): packages of task-specific knowledge and guidance loaded at inference time to steer agent behavior.

A central challenge, however, is that software engineering skills do not necessarily transfer across repositories. Recent work (Han et al., 2026) shows that most public software engineering skills provide little or no improvement on real-world repositories, and some can even hurt performance when their guidance does not fit the target context. This context dependence raises a fundamental question for skill learning: *which behaviors learned from past repositories will remain useful when the agent encounters a repository it has never seen before?*

Recent work has begun to learn skills directly from agent trajectories, allowing agents to reuse experience from previous tasks. Trace2Skill (Ni et al., 2026), for example, extracts trajectory-local lessons and hierarchically consolidates recurring ones into a shared skill directory; and, specifically for software engineering, STAIR (Xu et al., 2026) abstracts past repair trajectories into reusable actions and strategies that are retrieved and adapted for new issues. Yet observing a behavior in a successful trajectory does not by itself establish that the behavior is relevant to the successful outcome: it may be genuinely useful, merely incidental, or simply reflect a recurring habit of the

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.

backbone LLM. Moreover, even a useful behavior may depend on repository-specific conventions or workflows and fail to transfer elsewhere. This makes it difficult to determine which behaviors extracted from historical trajectories constitute genuinely reusable software engineering knowledge.

To address this challenge, we propose XRepoSkill, a trajectory-based skill learning method that separates the discovery of potentially useful behaviors from their empirical verification. We represent a skill as a collection of rules, each specifying what action to take and when to take it during issue resolution. XRepoSkill first contrasts successful and failed trajectories produced by the same agent on the same issue with different backbone LLMs. Because the issue, repository, and agent are fixed, their shared behavior provides little evidence for explaining the different outcomes; XRepoSkill therefore localizes where the two execution paths first diverge and uses the subsequent behavioral differences to derive candidate rules. This comparison narrows rule discovery to behaviors that distinguish successful from unsuccessful attempts rather than indiscriminately summarizing everything that appears in a successful trajectory.

Crucially, candidate rules do not enter the skill simply because they were discovered from a successful attempt. XRepoSkill associates each rule with an executable predicate that determines whether a trajectory exhibits the prescribed behavior, enabling systematic evaluation on trajectories beyond the original pair. Each rule is verified based on the association between its prescribed behavior and issue-resolution outcomes. XRepoSkill then examines whether rules from different repositories capture the same underlying behavior and consolidates them into repository-independent transferable rules. The resulting rules form the learned skill. At inference time, XRepoSkill selects rules relevant to the current issue, and provides them as additional guidance to the software engineering agent.

We evaluate XRepoSkill on the widely used SWE-bench Pro benchmark (Deng et al., 2026) and the more challenging long-horizon DeepSWE benchmark (Huang et al., 2026) using three backbone LLMs from different vendors. For skill learning, XRepoSkill uses publicly released trajectories from the official SWE-bench Verified leaderboard (SWE-bench, 2026). None of the repositories in either evaluation benchmark appears in this trajectory pool, ensuring that the evaluation directly measures transfer to previously unseen repositories. We use mini-SWE-agent (Yang et al., 2024) as the underlying software engineering agent, a lightweight yet strong open-source framework that has been widely adopted as a research baseline and as the foundation for several recent software engineering agents (Lee et al., 2026; Guo et al., 2026; Tripathy et al., 2026). We compare XRepoSkill with three recently proposed skill learning methods under the same agent and trajectory setting.

Across all six benchmark–LLM combinations, XRepoSkill consistently achieves the highest issue resolution rate. In particular, on the more challenging long-horizon DeepSWE benchmark, XRepoSkill improves issue resolution by 10.3 percentage points over the same agent without learned skills and by 5.0 points over the strongest skill learning baseline. The learned skill also transfers beyond the programming language represented during skill learning: although all source trajectories come from Python repositories, XRepoSkill achieves clear improvements on issues written in other programming languages. These results indicate that XRepoSkill distills transferable issue-resolution guidance rather than repository- or language-specific patterns.

In summary, this paper makes the following contributions:

- We introduce XRepoSkill, a trajectory-based skill learning approach that discovers candidate rules from behavioral divergences between successful and failed attempts, verifies them on independent trajectory evidence, and retains only behaviors recurring across repositories as transferable rules.
- We extensively evaluate XRepoSkill against recent skill learning methods on SWE-bench Pro and DeepSWE, showing the highest issue resolution rate in all evaluated benchmark–LLM combinations and strong transfer to previously unseen repositories and programming languages.
- We publicly release our code and data at https://github.com/XREPOSKILL/XRepoSkill.

2 METHODOLOGY

2.1 PROBLEM FORMULATION

We consider a software engineering agent A and a trajectory pool $\mathcal{T}$ that records its attempts to resolve historical software issues using different backbone LLMs. Each trajectory contains the agent's

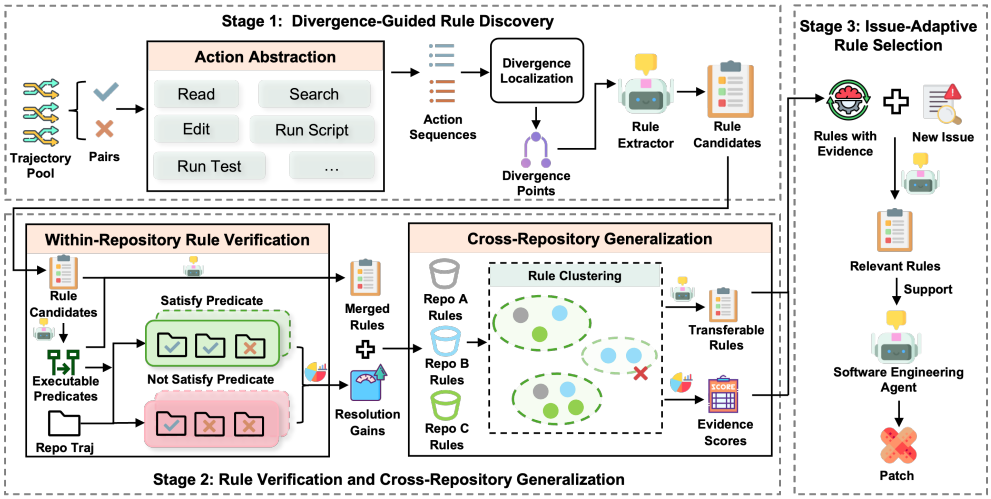

Figure 1: Overview of XRepoSkill.

sequence of actions and observations during one issue-resolution attempt, and $\mathcal{T}$ includes both successful and failed trajectories.

From $\mathcal{T}$, we seek to learn a transferable skill S. S consists of a collection of rules, each providing reusable guidance on what action to take and when to take it during issue resolution. We intend the learned skill to support future issues from repositories that do not appear in the trajectory pool.

For a new issue $i^\star$, XRepoSkill selects relevant rules from S and provides them to A as additional guidance. Our goal is for the resulting skill-guided agent to achieve a higher issue resolution rate on previously unseen repositories than the original agent without the learned skill.

2.2 XRepoSkill: In a Nutshell

Figure 1 presents an overview of XRepoSkill. Given the trajectory pool $\mathcal{T}$ of the software engineering agent A, XRepoSkill distills a transferable skill S and applies it to a new issue $i^\star$. XRepoSkill consists of three stages. First, **Divergence-Guided Rule Discovery** (Section 2.3) compares successful and failed trajectories of A on the same issue, identifies where their execution paths first diverge, and turns the divergent actions into candidate rules. Second, **Rule Verification and Cross-Repository Generalization** (Section 2.4) evaluates each candidate rule on other trajectories to determine whether trajectories that perform its prescribed action resolve issues more often than those that do not. It then consolidates rules from different repositories that capture the same underlying behavior into transferable rules, forming the skill S. Third, **Issue-Adaptive Rule Selection** (Section 2.5) selects rules relevant to $i^\star$ from S and provides them as guidance to the agent A.

2.3 Stage 1: Divergence-Guided Rule Discovery

The goal of this stage is to discover candidate rules from behavioral differences between successful and failed attempts on the same issue. For each issue in $\mathcal{T}$ with both outcomes, XRepoSkill pairs a successful trajectory with a failed trajectory produced by the same agent A under different backbone LLMs. Because the issue, repository, and agent are fixed, behaviors shared by the two trajectories cannot explain their different outcomes. XRepoSkill therefore aligns the trajectories, identifies where their execution paths first meaningfully diverge, and uses the subsequent behavioral differences to derive candidate rules. At this stage, these rules are only hypotheses; whether they are consistently associated with successful issue resolution is examined in Stage 2.

Action abstraction. It is unreliable to directly compare raw trajectories because the same behavior may be expressed through different tool calls. Thus, XRepoSkill maps each tool call to an abstract action defined by its *type* (e.g., reading a file, searching, editing, or running tests) and *target* (i.e., the object on which the call operates). For example, different commands that read the same file are

mapped to the same abstract action. Each trajectory is thereby transformed into an action sequence, allowing behaviorally equivalent steps to be aligned despite differences in their concrete realization.

Divergence localization. XRepoSkill aligns the two action sequences from the beginning and identifies the first point at which their behaviors meaningfully differ. Two actions match when they have the same type and target. Differences in the ordering of information-gathering actions, such as reading and searching, are tolerated because they may reflect different exploration orders rather than different resolution strategies. In contrast, an unmatched action that materially changes the subsequent course of the attempt, such as editing code or running a test, terminates the shared prefix. We define the first action after this shared prefix as the divergence point. The divergence point serves as an anchor for rule discovery: actions before it are shared by both the successful and failed attempts and therefore provide little evidence for explaining their different outcomes, whereas the behavior after it contains decisions unique to one execution path and thus provides a more informative search space for candidate rules. XRepoSkill retains the shared prefix in condensed form together with the two trajectory suffixes beginning at the divergence point for rule generation.

Candidate rule generation. The first divergence is not necessarily the behavior most relevant to the final outcome: once the trajectories separate, useful actions may occur later in the successful execution path. Rather than comparing the two suffixes step by step after the divergence point, XRepoSkill extracts candidate rules from three complementary reference points that remain well-defined without step-by-step alignment. The first examines the failed trajectory in light of the successful patch to identify information that may have helped the failed attempt. The second examines the successful trajectory to identify useful information-gathering and validation performed before editing. The third focuses on the divergence point itself and asks what the successful attempt does that the failed one does not. Each identified behavior is converted into a candidate rule specifying both *what action to take* and *when to take it*, and is phrased to apply beyond the specific issue from which it was extracted. The resulting candidate rules are passed to Stage 2.

2.4 Stage 2: Rule Verification and Cross-Repository Generalization

Stage 1 produces candidate rules from individual trajectory pairs, but a behavior that distinguishes one pair may be incidental to that pair or specific to its repository. Stage 2 therefore asks two questions: to what extent the behavior prescribed by a candidate rule is associated with successful issue resolution beyond the pair from which it is discovered, and whether the same behavior recurs across repositories. Rules capturing the same behavior across repositories are consolidated into the transferable skill S.

Executable rule predicates. Candidate rules are written in natural language, making repeated evaluation over extensive trajectories with LLM calls both expensive and potentially inconsistent. During candidate rule generation, XRepoSkill therefore produces an executable predicate together with each rule. The predicate is defined over the abstract actions from Stage 1 and captures the behavior prescribed by the rule, including whether an action occurs, whether one action precedes another, or how often an action occurs. For instance, the rule "run the failing test before editing any file" is represented by the predicate $\mathrm{before}(\mathrm{run_test}, \mathrm{first_edit})$. Evaluating the predicate on a trajectory requires no LLM call and deterministically indicates whether the trajectory exhibits the behavior described by the rule. Each rule thus partitions the trajectory pool $\mathcal{T}$ into trajectories that satisfy its predicate and those that do not, providing the basis for the verification that follows.

Within-repository rule verification. Different trajectory pairs from the same repository may yield candidate rules that describe the same behavior in different words. XRepoSkill therefore first merges, within each repository, candidate rules that describe the same behavior into one rule, using an LLM to consolidate their descriptions and select the predicate that best captures the shared behavior. XRepoSkill then verifies each merged rule by comparing the outcomes of trajectories that satisfy its predicate with those that do not. To avoid circular validation, this comparison excludes the successful–failed pairs from which any constituent rule was derived; otherwise, the rule could appear useful simply because it was extracted from a pair in which the successful trajectory exhibited the behavior and the failed trajectory did not.

The comparison is performed within issues rather than by pooling trajectories across issues. This controls for issue difficulty: a behavior may occur more often on easy issues and thus appear correlated with success even when it provides no benefit. For issue j, let n_{1j} and n_{0j} denote the numbers of trajectories that satisfy and do not satisfy the rule predicate, and let a_j and c_j denote the corresponding numbers of successful trajectories. We define the rule's *resolution gain* as

$$\text{gain} = \frac{\sum_j w_j \left(a_j/n_{1j} - c_j/n_{0j}\right)}{\sum_j w_j}, \qquad w_j = \frac{n_{1j} n_{0j}}{n_{1j} + n_{0j}}, \tag{1}$$

which is the Mantel–Haenszel estimator of a difference in rates across strata (Mantel & Haenszel, 1959). An issue contributes only when both groups are represented among its trajectories; otherwise, it provides no evidence for comparing them. Each rule also receives a z score, defined as its gain divided by its standard error (Sato et al., 1989), so that the strength of the evidence reflects both the estimated gain and the amount of supporting data. The gain and z score are not used as thresholds for retaining rules; instead, they inform issue-adaptive rule selection in Stage 3.

Cross-repository generalization. Evidence from a single repository is insufficient to establish transferability: a rule may depend on repository-specific conventions, workflows, or commands. XRepoSkill therefore requires the same behavior to be discovered and independently verified in multiple repositories before it enters the transferable skill.

To identify rules that capture the same behavior across repositories, XRepoSkill compares their executable predicates rather than their natural-language wording. Each predicate is evaluated on every trajectory in $\mathcal{T}$, turning each rule into a binary vector with one entry per trajectory that indicates whether the trajectory satisfies the predicate. Rules that prescribe the same behavior should therefore produce similar vectors. We measure this similarity using the ϕ correlation (Yule, 1912) between two binary vectors rather than raw agreement: because most predicates are false on most trajectories, two unrelated rules may agree frequently simply because both are absent, whereas correlation discounts such shared absence.

XRepoSkill groups rules using agglomerative clustering with distance $1 - \phi$ and complete linkage (Defays, 1977). This requires each pair of rules within a cluster to be sufficiently similar, preventing unrelated rules from being grouped together only through chains of intermediate rules. A cluster is retained only if it contains independently verified rules from at least two repositories. Its members are then consolidated by an LLM into a **transferable rule**, rewritten to remove repository-specific paths, symbols, and other identifiers. Clusters supported by only one repository are excluded.

A transferable rule may be supported with different strength across repositories. Let g_k and s_k denote the gain and standard error of the rule instance from repository k. XRepoSkill combines these estimates using inverse-variance weighting:

$$\bar{g} = \frac{\sum_k g_k/s_k^2}{\sum_k 1/s_k^2}, \qquad \bar{s} = \left(\sum_k 1/s_k^2\right)^{-1/2}, \tag{2}$$

so that more precise estimates receive greater weight (Cochran, 1954). The transferable rule receives gain $\bar{g}$ and z score $\bar{g}/\bar{s}$. The resulting skill S consists of these transferable rules together with their evidence scores, which are used in Stage 3 for issue-adaptive rule selection.

2.5 Stage 3: Issue-Adaptive Rule Selection

Stage 3 applies the learned skill S to a new issue $i^\star$ without modifying the underlying agent A. XRepoSkill changes only the additional guidance provided for the issue.

Rule selection. Given $i^\star$ and S, XRepoSkill asks the backbone LLM of A to select rules relevant to the issue. This selection is necessary because not every rule applies to every issue, and providing too many rules may dilute useful guidance. During selection, each rule is accompanied by its evidence, i.e., its gain and z score, which summarize the historical evidence supporting the rule and help the selector distinguish broadly supported guidance from weaker candidates.

Skill-guided resolution. The selected rules are provided to A as issue-specific guidance. The agent is instructed to verify each suggested action against the current repository before applying it,

so the rules guide rather than override its reasoning over the issue and codebase. It then resolves $i^\star$ using its original tools and execution procedure, with the selected rules as the only additional guidance introduced by XRepoSkill.

3 EVALUATION

3.1 RESEARCH QUESTIONS (RQS)

RQ1 (Effectiveness): How effective is XRepoSkill at improving issue resolution on previously unseen repositories?

RQ2 (Cross-Language Generalization): Can skills distilled by XRepoSkill generalize to issues in different programming languages?

RQ3 (Stability): How stable is the effectiveness of XRepoSkill across repeated runs?

RQ4 (Ablation Study): How do XRepoSkill's key design choices affect its effectiveness?

3.2 DATASETS AND EVALUATION METRIC

Trajectory pool used for skill learning. We use publicly released trajectories from the official SWE-bench Verified leaderboard (SWE-bench, 2026) as the source data for skill learning. These trajectories are generated by running mini-SWE-agent 2.0 (hereafter, mini-SWE-agent) on SWE-bench Verified (Jimenez et al., 2024), a widely used issue-resolution benchmark comprising 500 real-world, human-validated Python issues. mini-SWE-agent is a lightweight yet strong open-source agent that is widely adopted as a research baseline (Lee et al., 2026; Guo et al., 2026; Tripathy et al., 2026). The leaderboard includes trajectories produced with 12 backbone LLMs: Claude Haiku 4.5, Claude Sonnet 4.5, Claude Opus 4.5, Claude Opus 4.6, DeepSeek-V3.2, Gemini 3 Flash, Gemini 3 Pro, GLM-5, GPT-5.2, GPT-5 mini, Kimi K2.5, and MiniMax M2.5. Together, these runs yield a trajectory pool with 12 pass- or fail-labeled trajectories per issue.

Evaluation Benchmarks. We evaluate XRepoSkill on two recently introduced benchmarks, SWE-bench Pro (Deng et al., 2026) and DeepSWE (Huang et al., 2026).

- **SWE-bench Pro** contains 731 issue resolution tasks from 11 open-source repositories spanning Python, Go, JavaScript, and TypeScript. Due to budget constraints, we randomly sample 200 tasks for evaluation, comprising 69 Python, 73 Go, 52 JavaScript, and 6 TypeScript tasks.
- **DeepSWE** contains 113 long-horizon issue resolution tasks specifically designed for the benchmark rather than mined from public issue trackers, including 34 Python, 34 Go, 5 JavaScript, 35 TypeScript, and 5 Rust tasks. We evaluate on all 113 tasks.

We have confirmed that none of the repositories in either benchmark appears in our trajectory pool, ensuring that the evaluation measures transfer to previously unseen repositories.

Evaluation Metric. We use the resolution rate (**%Resolved**) as the evaluation metric, defined as the proportion of benchmark issues resolved by an agent.

3.3 BASELINES AND LLMS

Besides the raw mini-SWE-agent without any skills, we compare XRepoSkill with three representative trajectory-based skill-learning methods:

- **SkillOpt** (Yang et al., 2026) treats the skill document as a trainable state of an agent and iteratively updates it from scored trajectories, retaining only changes that improve held-out performance.
- **STAIR** (Xu et al., 2026) abstracts past repair trajectories into reusable actions and strategies, then retrieves and adapts relevant experience into an issue-specific plan for each new task.
- **Trace2Skill** (Ni et al., 2026) distills trajectory-local lessons in parallel and hierarchically consolidates them into a shared transferable skill directory used at inference time.

Table 1: (RQ1) Issue resolution rates (%) of different methods across LLMs on SWE-bench Pro and DeepSWE. Avg. denotes the mean over the three LLMs for each benchmark. The highest rate in each column is shown in bold.

	SWE-bench Pro				DeepSWE			
Method	DeepSeek	GPT	MiMo	Avg.	DeepSeek	GPT	MiMo	Avg.
mini-SWE-agent	51.5	50.5	41.5	47.8	46.0	44.2	11.5	33.9
SkillOpt	54.0	51.5	41.0	48.8	40.7	39.8	13.3	31.3
STAIR	**55.5**	51.0	48.5	51.7	43.4	46.0	14.2	34.5
Trace2Skill	55.0	52.0	45.0	50.7	49.6	53.1	15.0	39.2
XRepoSkill	**55.5**	**56.0**	**50.0**	**53.8**	**58.4**	**56.6**	**17.7**	**44.2**

To ensure a fair comparison, we use the same LLM, DeepSeek-V4-Pro-0813, for XRepoSkill and all baseline methods to learn skills from the same trajectory pool.

For issue resolution, we run all methods with the same mini-SWE-agent and evaluate each with three backbone LLMs from different vendors: DeepSeek-V4-Flash-0731, GPT-5.6-Luna, and MiMo-V2.5-Pro, hereafter referred to as **DeepSeek**, **GPT**, and **MiMo**. We set the reasoning effort to high for all three models, as issue resolution requires reasoning over the issue description, repository context, and tool outputs before making code edits.

4 Results and Analysis

4.1 RQ1: Effectiveness

RQ1 evaluates the effectiveness of XRepoSkill in resolving issues from previously unseen repositories. Table 1 reports the resolution rates of XRepoSkill and the baselines across three backbone LLMs on SWE-bench Pro and DeepSWE. XRepoSkill achieves the highest resolution rate in all six benchmark–LLM combinations. This consistency is notable because the strongest baseline varies across benchmarks: on average, STAIR performs best on SWE-bench Pro, whereas Trace2Skill is strongest on DeepSWE. In contrast, XRepoSkill remains the top-performing method across both benchmarks and all three LLMs, demonstrating more consistent effectiveness than existing skill-learning methods.

On SWE-bench Pro, XRepoSkill achieves an average resolution rate of 53.8%, outperforming mini-SWE-agent and the strongest baseline, STAIR, by 6.0 and 2.1 percentage points, respectively. The margin is larger on the more challenging long-horizon DeepSWE, where XRepoSkill reaches 44.2%, exceeding mini-SWE-agent and the strongest baseline, Trace2Skill, by 10.3 and 5.0 points, respectively. This larger gain suggests that XRepoSkill is particularly effective on more demanding tasks.

We also compare the runtime cost of each method. The costs are broadly comparable across methods. On SWE-bench Pro, XRepoSkill costs \$0.11 per issue on average, close to mini-SWE-agent (\$0.09), SkillOpt (\$0.12), STAIR (\$0.10), and Trace2Skill (\$0.10). On DeepSWE, XRepoSkill costs \$0.25 per issue, compared with \$0.22 for mini-SWE-agent, SkillOpt, and STAIR, and \$0.18 for Trace2Skill. Thus, XRepoSkill operates at a similar cost scale to existing methods while achieving the highest resolution rate in all six benchmark–backbone combinations, indicating a favorable effectiveness–cost trade-off.

4.2 RQ2: Cross-Language Generalization

RQ2 examines whether the skills learned by XRepoSkill from Python-only repositories generalize to other programming languages. Since RQ1 identifies STAIR and Trace2Skill as the strongest baselines on SWE-bench Pro and DeepSWE, respectively, Table 2 compares XRepoSkill with both and reports the resolution-rate gains for each programming language, averaged over the three backbone LLMs. The results show that XRepoSkill remains competitive on Python while achieving clearer advantages on languages absent from the skill-learning corpus, suggesting that its learned rules transfer beyond the language from which they are distilled.

Table 2: (RQ2) Resolution-rate gains (%) of XRepoSkill over STAIR and Trace2Skill across programming languages, averaged over three backbone LLMs. # denotes the number of issues for each language in each subset.

Language	SWE-bench Pro			DeepSWE		
	#	vs. STAIR	vs. Trace2Skill	#	vs. STAIR	vs. Trace2Skill
Python	69	−1.0	+1.0	34	+5.9	+0.0
Go	73	+2.7	+4.1	34	+2.0	+3.9
JavaScript	52	+6.4	+4.5	5	+13.3	+0.0
TypeScript	6	−5.6	+5.6	35	+21.0	+10.5
Rust	–	–	–	5	+6.7	+13.3

On Python, the only language represented in the trajectory pool, XRepoSkill performs similarly to the strongest baselines. On SWE-bench Pro, it trails STAIR by 1.0 percentage point and exceeds Trace2Skill by 1.0 point; on DeepSWE, it outperforms STAIR by 5.9 points and matches Trace2Skill. The advantage becomes more consistent once the programming language changes. On Go, XRepoSkill outperforms both baselines on both benchmarks, by 2.0–4.1 points. On JavaScript in SWE-bench Pro, it exceeds STAIR and Trace2Skill by 6.4 and 4.5 points, respectively. On TypeScript in DeepSWE, the gains further increase to 21.0 and 10.5 points. Together, these results suggest that XRepoSkill distills issue-resolution procedures that generalize across programming languages rather than relying primarily on Python-specific patterns. The results for TypeScript on SWE-bench Pro and for JavaScript and Rust on DeepSWE should be interpreted with caution, as these subsets contain only 6, 5, and 5 issues, respectively.

4.3 RQ3: Stability

RQ3 examines whether the effectiveness of XRepoSkill remains stable across repeated runs. In this RQ, we focus on DeepSWE, whose long-horizon tasks make repeated-run stability particularly relevant, and compare XRepoSkill with the raw mini-SWE-agent without learned skills and with Trace2Skill, the strongest baseline on DeepSWE in RQ1. Given the computational cost of repeated evaluation and our limited budget, we use DeepSeek as the backbone LLM. Following the DeepSWE evaluation protocol, we run each method four times on all tasks and report the mean resolution rate with its 95% confidence interval. We further report Pass@4 and All@4, the proportions of issues resolved in at least one and in all four runs, respectively.

Table 3: (RQ3) Issue resolution results over four runs on DeepSWE with DeepSeek. Mean denotes the mean resolution rate (%) across the four runs, with 95% confidence intervals. Pass@4 and All@4 denote the proportions (%) of issues resolved in at least one run and in all four runs, respectively.

Method	Mean	Pass@4	All@4
mini-SWE-agent	45.6 ± 2.50	77.9	17.7
Trace2Skill	49.3 ± 2.69	81.4	21.2
XRepoSkill	**58.6 ± 2.49**	**89.4**	**32.7**

Table 3 reports the results. XRepoSkill maintains a clear advantage across repeated runs, achieving a mean resolution rate of 58.6%, 13.0 and 9.3 percentage points higher than mini-SWE-agent and Trace2Skill, respectively. Its 95% confidence interval does not overlap with either baseline. XRepoSkill also achieves the highest Pass@4 (89.4%) and All@4 (32.7%), exceeding Trace2Skill by 8.0 and 11.5 points, respectively. These results show that the gains observed in RQ1 persist across repeated runs rather than arising from a favorable single execution.

4.4 RQ4: Ablation Study

RQ4 evaluates the key design choices in XRepoSkill. As in RQ3, we conduct the ablation study on DeepSWE with DeepSeek as the backbone LLM. We construct five variants spanning rule discovery, within-repository verification, cross-repository generalization, and rule selection. Each variant modifies only the designated component while keeping the rest of the pipeline unchanged: (1) **w/ Trace2Skill** replaces the candidate rules discovered in Stage 1 with rules generated by Trace2Skill; (2) **w/o within-repo** removes within-repository rule verification and directly passes rules to cross-repository consolidation; (3) **w/o cross-repo** removes cross-repository consolidation and retains the independently verified repository-level rules; (4) **w/ all rules** provides the

full rule set to the agent instead of selecting relevant rules in Stage 3; and (5) **w/o evidence** removes the historical evidence associated with each rule from the Stage 3 selection process.

Table 4: (RQ4) Issue resolution rates (%) of XRepoSkill and its ablation variants on DeepSWE with DeepSeek.

Method	%Resolved
XRepoSkill	**58.4**
w/ Trace2Skill	54.9
w/o within-repo	53.1
w/o cross-repo	51.3
w/ all rules	48.7
w/o evidence	55.8

Table 4 reports the results. Each ablation variant degrades the performance of XRepoSkill, suggesting that its gains do not stem from a single component but from the combined contributions of rule discovery, verification, cross-repository generalization, and rule selection. The largest drop comes from replacing issue-adaptive selection with all rules (58.4% $\rightarrow$ 48.7%), followed by removing cross-repository generalization (51.3%) and removing within-repository verification (53.1%). These results highlight the importance of verifying rules and consolidating behaviors recurring across repositories before inference and selecting only rules relevant to the current issue. Replacing divergence-guided discovery with Trace2Skill reduces the resolution rate to 54.9%, suggesting that the source of candidate rules also matters, while removing rule evidence lowers performance to 55.8%, suggesting that historical support helps the selector prioritize useful guidance.

5 Related Work

Software engineering agents. Software engineering agents primarily target repository-level issue resolution. SWE-bench (Jimenez et al., 2024) established this setting as a standard benchmark and spurred rapid progress in systems such as SWE-agent (Yang et al., 2024), OpenHands (Wang et al., 2025a), AutoCodeRover (Zhang et al., 2024), and Agentless (Xia et al., 2025a). mini-SWE-agent provides a lightweight, shell-based implementation of this workflow and has become a widely used open-source research baseline, with several recent agents building on it (Guo et al., 2026; Xia et al., 2025b). Subsequent work has further improved issue-resolution performance through stronger search and planning, multi-agent collaboration, and test-time scaling (Tao et al., 2024; Zhang et al., 2025; Gao et al., 2025; Ehrlich et al., 2025). Our work studies skill learning in the same setting using mini-SWE-agent as the underlying agent, while keeping the agent itself unchanged so that performance differences reflect the skills rather than changes to the agent architecture.

Skill learning for agents. Skills provide agents with reusable task guidance at inference time. Their effectiveness, however, depends strongly on how they are constructed (Han et al., 2026). A growing body of work distills reusable skills from agent trajectories, including executable programs, natural-language experience, workflows, and skill documents (Wang et al., 2024; Zheng et al., 2025; Zhao et al., 2024; Ouyang et al., 2026; Wang et al., 2025b). Among recent skill-learning methods, SkillOpt (Yang et al., 2026) optimizes skills against validation performance, while Trace2Skill (Ni et al., 2026) consolidates recurring trajectory-local lessons into transferable skills. In software engineering specifically, STAIR (Xu et al., 2026) abstracts repair trajectories into reusable plans for new issues. XRepoSkill differs by explicitly verifying candidate rules on trajectories beyond those from which they are discovered and retaining them as transferable only when the same behavior recurs across repositories.

6 Conclusion

This paper studies how to learn software engineering skills that remain useful beyond the repositories from which they are distilled. We introduce XRepoSkill, a trajectory-based skill learning approach that contrasts successful and failed attempts to discover candidate rules, verifies their prescribed behaviors on independent trajectories through executable predicates, and retains only rules independently verified across repositories as transferable guidance. Evaluated on SWE-bench Pro and DeepSWE with three backbone LLMs, XRepoSkill consistently achieves the highest issue resolution rate in all six benchmark–LLM combinations, compared with both the underlying agent without learned skills and recent skill-learning baselines. These results highlight the value of going beyond extracting recurring behaviors from past experience by testing candidate behaviors against broader trajectory evidence and validating them beyond their original repository context.

AI USE STATEMENT

Generative AI tools are used in this work in three ways. First, LLMs are an integral part of the proposed method and experimental pipeline. Second, we use generative AI tools to assist with code implementation. Third, we use generative AI tools to improve the clarity, grammar, and presentation of the manuscript.

All AI-assisted code is reviewed by the authors and tested as part of the experimental pipeline. All AI-assisted writing is manually reviewed, revised, and verified by the authors. The authors design the research methodology, conduct and validate the experiments, analyze the results, and determine the scientific claims and conclusions. We take responsibility for the final content of this work, including all text, code, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To support reproducibility, we have released the source code, data, and skill corpus used in this work at https://github.com/XREPOSKILL/XRepoSkill.

REFERENCES

Anthropic. Agent Skills. `https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview`, 2026. Accessed September 18, 2026.

William G. Cochran. The combination of estimates from different experiments. *Biometrics*, 10(1): 101–129, 1954.

Daniel Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4): 364–366, 1977.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026.

Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. CodeMonkeys: Scaling test-time compute for software engineering. *arXiv preprint arXiv:2501.14723*, 2025.

Pengfei Gao, Zhao Tian, Xiangxin Meng, Xinchen Wang, Ruida Hu, Yuanan Xiao, Yizhou Liu, Zhao Zhang, Junjie Chen, Cuiyun Gao, Yun Lin, Yingfei Xiong, Chao Peng, and Xia Liu. Trae Agent: An LLM-based agent for software engineering with test-time scaling. *arXiv preprint arXiv:2507.23370*, 2025.

Yaoqi Guo, Yang Liu, Jie M. Zhang, Yun Ma, Yiling Lou, and Zhenpeng Chen. SWE-Doctor: Guiding software engineering agents with runtime diagnosis from multi-faceted bug reproduction tests. *arXiv preprint arXiv:2607.00990*, 2026.

Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. SWE-Skills-Bench: Do agent skills actually help in real-world software engineering? *arXiv preprint arXiv:2603.15401*, 2026.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring frontier coding agents on original, long-horizon engineering tasks. *arXiv preprint arXiv:2607.07946*, 2026.

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

Hyunji Lee, Minseon Kim, Chinmay Singh, Matheus Pereira, Atharv Sonwane, Isadora White, Elias Stengel-Eskin, Mohit Bansal, Zhengyan Shi, Alessandro Sordoni, Marc-Alexandre Côté, Xingdi Yuan, and Lucas Caccia. Gistify: Codebase-level understanding via runtime execution. In *International Conference on Learning Representations (ICLR)*, 2026.

Nathan Mantel and William Haenszel. Statistical aspects of the analysis of data from retrospective studies of disease. *Journal of the National Cancer Institute*, 22(4):719–748, 1959.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations (ICLR)*, 2026.

Tosiya Sato, Sander Greenland, and James M. Robins. On the variance estimator for the Mantel–Haenszel risk difference. *Biometrics*, 45(4):1323–1324, 1989.

SWE-bench. SWE-bench leaderboards. `https://www.swebench.com/`, 2026. Accessed September 18, 2026.

Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. MAGIS: LLM-based multi-agent framework for GitHub issue resolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Arihant Tripathy, Ch Pavan Harshit, and Karthik Vaidhyanathan. SWEnergy: An empirical study on energy efficiency in agentic issue resolution frameworks with SLMs. In *Proceedings of the 2026 International Workshop on Agentic Engineering (AGENT '26)*, pp. 104–111. ACM, 2026. doi: 10.1145/3786167.3788406.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

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, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *PMLR*, pp. 63897–63911, 2025b.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering (PACMSE)*, 2 (FSE):801–824, 2025a. doi: 10.1145/3715754.

Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. Live-SWE-agent: Can software engineering agents self-evolve on the fly? *arXiv preprint arXiv:2511.13646*, 2025b.

Yisen Xu, Jiayuan Zhou, Ruiqi Pan, and Tse-Hsun Chen. Reusing past repairs through hierarchical trajectory abstraction for coding agents. *arXiv preprint arXiv:2607.29658*, 2026.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

G. Udny Yule. On the methods of measuring association between two attributes. *Journal of the Royal Statistical Society*, 75(6):579–652, 1912.

Kexun Zhang, Weiran Yao, Zuxin Liu, Yihao Feng, Zhiwei Liu, Rithesh Murthy, Tian Lan, Lei Li, Renze Lou, Jiacheng Xu, Bo Pang, Yingbo Zhou, Shelby Heinecke, Silvio Savarese, Huan Wang, and Caiming Xiong. Diversity empowers intelligence: Integrating expertise of software engineering agents. In *International Conference on Learning Representations (ICLR)*, 2025.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pp. 1592–1604, 2024. doi: 10.1145/3650212.3680384.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2024.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

A Implementation Details

Section 2 describes the design of XRepoSkill. This appendix lists, stage by stage, the parameters, prompt constraints, and formats with which the design is implemented, using the terms of Section 2. Every LLM call in Stage 1 and Stage 2 is made to DeepSeek-V4-Pro-0813, the LLM that XRepoSkill uses for skill learning in Section 3; we refer to it below as the skill-learning LLM. The two calls of Stage 3 are made to the backbone LLM of the agent A.

A.1 Stage 1: Divergence-Guided Rule Discovery

Pairing. Backbone LLMs are ranked by their resolution rate on the trajectory pool. On each issue with both successful and failed trajectories, every failed trajectory is combined with every successful trajectory from a backbone LLM with a higher resolution rate, the combinations are ranked by the gap in resolution rate between the two LLMs, and the top four become pairs. A check before pairing asserts that no issue of the pool appears in either evaluation benchmark.

Action abstraction. Before a tool call is mapped to an abstract action, compound commands joined by `&&` or `;` are split into one command each, and leading `cd`, environment variable assignments, and `timeout` prefixes are stripped. A bare `cd`, `pwd`, or `export` yields no action. The type of the action is then decided by the program that leads the command. A read prints part of a file with `cat`, `head`, `tail`, `less`, `nl`, `sed -n`, or `awk`; its target is the first file path in the command. A search runs `grep`, `rg`, `ack`, `find`, `ls`, `which`, `locate`, or `tree`; its target is the quoted pattern, or the first non-option argument when nothing is quoted. An edit runs `sed -i`, redirects the output of `cat`, `echo`, `printf`, or `tee` into a source or configuration file, or runs `patch` or `git apply`; its target is the written file. A test run invokes `pytest`, `tox`, `python -m pytest`, `python -m unittest`, or a `runtests.py` script; its target is the first argument containing "test". A script run invokes `python` or `ipython` for anything other than a test run; its target is the first file path. An install runs `pip`, `conda`, or `apt`; its target is the package name. A git action is any other `git` subcommand, with the subcommand as target. A submission is the final-output marker of mini-SWE-agent. Every remaining command is of type other, with an empty target. A regular expression in a predicate is matched case-insensitively against the full command text before splitting and stripping.

Divergence localization. The shared prefix is extended across matching blocks of the two action sequences. A gap of unmatched actions between two matching blocks is tolerated only if it holds at most three actions on each side and none of them is an edit, a test run, a script run, an install, or a submission. The text handed to the skill-learning LLM contains the shared prefix as a compact list of actions, the two trajectory suffixes from the divergence point onward within a fixed character budget, a compact list of the actions that follow the budget, and the position of every command in its trajectory, so that a prompt can cite a command by position.

Candidate rule generation. Each pair is read by three prompts, one per reference point of Section 2.3, and each prompt sees the text described above and nothing else from the trajectories. The first prompt reads the failed suffix together with the patch that the successful trajectory submitted and reports the earliest step that the failed trajectory skipped. The second reads the successful suffix only and reports up to three steps taken before its first edit. The third reads both suffixes and reports the earliest step that only the successful trajectory took, citing one command position on each side. The first and third prompts return one candidate rule each and the second returns one to three, so each pair yields three to five candidate rules. Each candidate rule is returned as a JSON object with a paragraph of at most 1,000 characters, a category, one of lookup, verification, replay, or search recipe, the trajectory identifiers and command positions it cites as evidence, and its executable predicate. A candidate rule that states a fact about the code instead of prescribing an action carries no predicate and must name at least one real file path or symbol; such rules cannot be verified in Stage 2 and never enter the transferable skill.

Rejection. A candidate rule is rejected if its paragraph exceeds the length limit, if it prescribes an action and contains a file path followed by a line number or three or more rare symbols such as double-underscore method names, if its predicate does not parse, or if its predicate is true on fewer than 10% or more than 90% of the trajectories of the pool. A rejected rule is re-prompted once with the reason and dropped if it fails again.

A.2 Stage 2: Rule Verification and Cross-Repository Generalization

Within-repository merging. Within a repository, candidate rules of the same category are handed to the skill-learning LLM in batches, and each batch is merged into between 3 and 10 rules; rules with the same title from different batches are merged again. The number of candidate rules merged into a rule is its support, and the predicate is also selected by the LLM. The prompt requires that exact commands, search patterns, and paths present in the candidate rules be kept verbatim, and rejects a merged rule reduced to advice such as "reproduce first". A screen by the skill-learning LLM drops a merged rule only if it neither prescribes an action nor states a fact anchored to a file path or symbol, or if it contradicts conventions already recorded for its repository. An n-gram audit flags any rule whose text shares six or more consecutive tokens with the reference patch of one of its source issues.

Resolution gain and z score. The weight w_j in equation 1 is largest when both groups of issue j are large and is zero when every trajectory of the issue falls in one group. The standard error of the gain is the estimator of Sato et al. (1989). Within each repository, rules are ordered by z score. A rule whose predicate takes the same value on every trajectory of its repository receives no gain; such rules and rules that state facts are placed at the tail of the order, ordered by support.

Cross-repository generalization. The correlation ϕ is computed only for verified rules. The cutoff on ϕ is read from the histogram of ϕ over all pairs of rules, which has a spike near one and a broader mass at lower values; the cutoff is the local minimum between the two. When the rules of a cluster are rewritten into a transferable rule, at most one verbatim example command per repository is kept, and rules that give different advice are listed as outliers. In equation 2, the standard error s_k of each merged rule is its gain divided by its z score, and the support of the transferable rule is the sum of the supports of the rules it merges.

Skill file. The skill S is written as one Markdown file. Each transferable rule is one bullet holding its title, a tag, and its body, and the bullets are ordered by z score. The tag holds the gain of the

rule and its z score, for example "(gain +5.7pp, z +15.59)". Rules whose cluster came from a single repository do not enter S.

A.3 Stage 3: Issue-Adaptive Rule Selection

Rule selection. Selection uses two calls to the backbone LLM of A. The first call receives the issue description, cut to 6,000 characters, and the titles of all rules, and returns a summary of the bug in two to three sentences and a list of the kinds of guidance that would help. The second call receives the issue description, that summary, and every rule with its identifier, tag, title, and body in z score order, and returns at most three rule identifiers with a one-line reason each. The system prompt of the second call asks for the rules most likely to help an engineer fix the bug. Identifiers that do not exist are discarded, and the selection may be empty. In our experiments, the rules shown to the two calls are the 36 transferable rules of S; Appendix C reports which of them were chosen.

Skill-guided resolution. The selected rules are placed above the issue description in the first user message of mini-SWE-agent, with the framing that they are guidance on how to investigate and acquire facts at runtime and that each step must be verified against the actual code before acting. When the selection is empty, the agent runs without added text. The agent makes no further calls for rules during its loop, and the summary, the reasons, and the tags never enter its prompt.

B Details of the Evaluation

B.1 Datasets

The trajectory pool of Section 3 is downloaded from the public SWE-bench experiments repository. Its 500 issues come from 12 Python repositories, which are the repositories within which Stage 2 verifies rules and across which it merges them. The 200 SWE-bench Pro instances of the main results cover all 11 repositories of the public set of the benchmark. Each DeepSWE task specifies a change to a repository in natural language and is graded by hidden tests.

B.2 Evaluation Protocol

All methods run the same mini-SWE-agent with the same system prompt, tool interface, and step and cost limits, and each method changes only the text placed before the issue description in the first user message. The skills of the three baselines are learned from the same trajectory pool with the skill-learning LLM of Appendix A. The per-issue plan calls of STAIR and the two selection calls of XRepoSkill use the backbone LLM that runs the repair. Except in the stability study of Section 4.3, each issue is run once per method and backbone LLM, and the reported resolution rates come from that single run.

SWE-bench Pro issues are mined from public repositories, and the commit that fixed each issue is present in the public history of the repository, so an agent that can read that history or reach the network can copy the fix. We therefore build the container of every SWE-bench Pro issue in three steps. First, the repository is checked out at the base commit that the issue specifies. Second, the `.git` directory is deleted after the checkout, so `git log`, `git diff`, `git show`, and every other command that reads the history fail, and the agent sees only the files of the working tree. Third, the container is started without network access, so `git fetch`, `curl`, `pip install`, and any other command that reaches outside the container fail. The official evaluation images keep the history and the network, so resolution rates measured in them are not directly comparable to ours. Every method and backbone LLM in the paper runs in the same containers, so the comparison between methods is unaffected. DeepSWE tasks are authored for the benchmark rather than mined from public repositories, so no upstream fix exists to be found, and we run them in the official environment.

B.3 Cost Accounting

The cost per issue reported in Section 4.1 is computed from the token counts recorded in the trajectory of each run and in the records of the calls made before the repair loop starts, namely the two

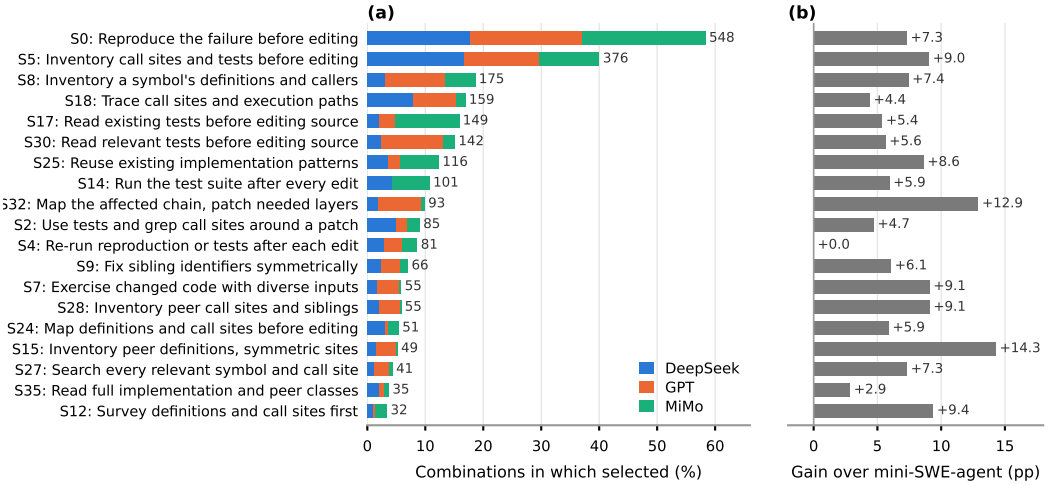

Figure 2: The 19 transferable rules selected in at least 30 of the 939 combinations of an issue and a backbone LLM in the main results, ordered by selection count. (a) Share of combinations in which the rule was selected, stacked by backbone LLM; the number at the end of a bar is the count. (b) Resolution rate of XRepoSkill minus that of mini-SWE-agent, in percentage points, on the combinations in which the rule was selected.

selection calls of XRepoSkill and the three plan calls of STAIR. Every call is priced with the list price of its provider, in US dollars per million tokens. Input tokens are split into cache hits and cache misses, and each part is billed at the rate the provider charges for it: 0.003 and 0.15 for DeepSeek at the off-peak rate, 0.02 and 0.20 for GPT, whose cache writes are billed at 0.25, and 0.0036 and 0.435 for MiMo. Output tokens are billed at 0.60, 1.20, and 0.87, respectively. Table 5 lists the resulting cost per issue for every method, backbone LLM, and benchmark; the averages over backbone LLMs are the numbers quoted in Section 4.1. The two selection calls of XRepoSkill cost $0.01 per issue with DeepSeek and GPT and $0.03 with MiMo, and the rest of its cost comes from the repair loop.

Table 5: Cost per issue in US dollars for every method, backbone LLM, and benchmark of the main results. Avg. is the mean over the three backbone LLMs on the same benchmark.

	SWE-bench Pro				DeepSWE			
Method	DeepSeek	GPT	MiMo	Avg.	DeepSeek	GPT	MiMo	Avg.
mini-SWE-agent	0.133	0.089	0.034	0.086	0.150	0.390	0.128	0.223
SkillOpt	0.167	0.148	0.041	0.119	0.135	0.407	0.125	0.222
STAIR	0.140	0.098	0.052	0.097	0.150	0.358	0.140	0.216
Trace2Skill	0.153	0.117	0.036	0.102	0.139	0.274	0.124	0.179
XRepoSkill	0.156	0.110	0.065	0.110	0.161	0.426	0.152	0.246

C WHICH RULES ARE SELECTED, AND WITH WHAT RESULT

The main results place at most three rules in the prompt of each issue, and the second selection call of Stage 3 decides which. This section examines the selections made for the 939 combinations of an issue and a backbone LLM in the main results, recovered from the prompt of each run, and relates every transferable rule to the outcome on the combinations in which it was selected.

Selection concentrates on a few rules. Figure 2(a) shows the 19 transferable rules selected in at least 30 combinations. S0, reproduce the reported failure before touching source, was selected in 58.4% of the combinations and S5, inventory call sites and tests before modifying shared behavior, in 40.0%. The five most selected rules account for half of all selections and the top 11 for 80%.

Of the other 17 rules, four were never selected and eight were selected fewer than ten times, so a third of the skill went almost unused. Part of this tail consists of rules that prescribe the same action as a frequent one in different words, for example, to read the tests before editing. Cross-repository generalization kept them apart because their predicates are true on different trajectories, and the selection call spreads its choices over them.

Which rules are selected depends on the backbone LLM that selects. MiMo selected S17 in 33% of its combinations against 6% for DeepSeek and 8% for GPT. GPT selected S30 in 32% of its combinations and never selected S14. DeepSeek selected S5 in 50% of its combinations. The Spearman rank correlation of selection counts between two backbone LLMs lies between 0.68 and 0.81, whereas between the two benchmarks, with backbone LLMs pooled, it is 0.93.

The agent resolves more issues when a rule is selected. Figure 2(b) shows, for the same 19 rules, the resolution rate of XRepoSkill minus that of mini-SWE-agent on the combinations in which the rule was selected. The difference is positive for 18 rules and zero for one, S4, and ranges from 2.9 to 14.3 percentage points; the two most selected rules, S0 and S5, gain 7.3 and 9.0 points on 548 and 376 combinations. Because up to three rules are placed in the prompt together and the combinations are chosen by the selection call rather than at random, a gain belongs to the selected set rather than to one rule.