BEYOND REJECTION SAMPLING: LEARNING FROM GUIDED ROLLOUTS TO IMPROVE LLMS FOR SOFTWARE ENGINEERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Rejection-sampling supervised fine-tuning (SFT) for coding agents retains only trajectories that pass executable tests. On difficult repository tasks, repeated unguided rollouts often produce no successful trajectory. Consequently, **many available tasks and execution environments yield no retained SFT examples**, wasting executable software-engineering data. We use answer-derived guidance to recover training trajectories. Based on when guidance enters an interactive rollout, we design three mechanisms: **Pre guidance** derives a hint from the reference solution before initial rollout; **Post guidance** translates an unsuccessful attempt and the reference solution into a hint for fresh retry; and **Intra guidance** provides answer-derived assistance at eligible interaction steps. Our method, *SFT with Guided-Success Augmentation* (GS-SFT), augments rejection-sampling SFT with verifier-accepted guided trajectories and uses ordinary supervised fine-tuning. We distinguish **Supply**, retained SFT examples normalized by the number of executable tasks, from **Utility**, the task resolved rate after SFT, measured on SWE-bench Verified and SWE-bench Multilingual. Across the ordered guidance constructions, Supply increases steadily, whereas **Utility peaks at an earlier or intermediate non-oracle condition and then decreases under more explicit guidance**. Direct reference patches produce the most data but not the highest resolved rate. For offline guidance, our results favor retaining hints in the SFT input rather than removing them. The additional guided data produces the largest resolved-rate gains on issues with no actionable solution or reproduction lead, while also improving issues that already identify a likely solution or code location. Overall, moderately informative answer-derived guidance provides a simple way to use executable software-engineering data more efficiently during SFT.

1 INTRODUCTION

Large language models have evolved from code-completion systems into *coding agents* that inspect repositories, edit files, and execute tests. SWE-bench formalized a representative repository-level task: given a real issue and repository snapshot, an agent must produce a patch that passes the project test harness (Jimenez et al., 2024). This setting requires sustained interaction to locate files, understand behavior, modify code, and validate the resulting patch. SWE-agent and OpenHands therefore treat software engineering as interaction between an agent and its environment (Yang et al., 2024; Wang et al., 2025).

Executable environments also provide a natural training signal. Coding-agent training samples trajectories, executes their patches, retains successes, and applies supervised fine-tuning (SFT). SWE-Gym, R2E-Gym, and SWE-smith scale this pattern through executable environments, synthetic task construction, and verified trajectories (Pan et al., 2025; Jain et al., 2025; Yang et al., 2025b).

Data-utilization bottleneck. On difficult tasks, repeated unguided rollouts often fail to produce any test-passing trajectory. These tasks have functioning environments and reference solutions, yet contribute no SFT example. The challenge is to convert executable tasks into verified training data.

Our approach. We use answer-derived guidance to make these executable tasks productive. The rollout timeline gives three intervention points. **Pre guidance** derives a hint from the reference

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.

solution before the initial rollout. **Post guidance** translates an unsuccessful attempt and the reference solution into a hint for a fresh retry; the retry agent receives the hint but not the raw failed history. **Intra guidance** supplies answer-derived assistance at eligible steps during the rollout and may refine the agent response. Our method, *SFT with Guided-Success Augmentation* (GS-SFT), augments rejection-sampling SFT with verifier-accepted guided trajectories while retaining ordinary supervised training and guidance-free evaluation.

Supply and Utility. Supply is retained SFT examples normalized by the number of executable tasks. **Utility** is the task resolved rate after SFT on SWE-bench Verified and SWE-bench Multilingual. Together, they measure data production and downstream coding-agent capability.

Findings. Supply increases steadily across the ordered guidance constructions, but Utility does not. Earlier or intermediate non-oracle conditions achieve the highest resolved rates, while direct reference patches produce the most data without producing the strongest model. For offline Pre/Post guidance, retaining the hint in the SFT input performs better than removing it from the trajectory it conditioned. The additional guided data yields its largest gains on issues with no actionable solution or reproduction lead and also improves issues that already identify a likely solution or code location.

Contributions. First, we formulate three rollout-aligned mechanisms for deriving successful coding-agent trajectories from reference information. Second, we introduce the distinction between Supply and Utility for evaluating guided trajectory collection. Third, our systematic study shows that moderately informative guidance can improve both verified-data production and downstream task resolution, providing a simple way to use executable software-engineering data more efficiently during SFT.

2 Related Work

From code generation to repository-level evaluation. Early execution-based benchmarks evaluate self-contained synthesis, including HumanEval and APPS (Chen et al., 2021; Hendrycks et al., 2021). RepoCoder, RepoBench, and CrossCodeEval introduced repository-level retrieval and cross-file context (Zhang et al., 2023; Liu et al., 2024; Ding et al., 2023). SWE-bench evaluates issue resolution through executable repository patches (Jimenez et al., 2024); later benchmarks extend this setting to visual, multilingual, and freelance tasks (Yang et al., 2025a; Zan et al., 2025; Rashid et al., 2025; Miserendino et al., 2025). SWE-bench Live and SWE-rebench focus on freshness and reproducible task construction (Zhang et al., 2025; Badertdinov et al., 2025).

Coding-agent architectures. ReAct interleaves reasoning with environment actions, and CodeAct represents actions as executable code (Yao et al., 2023; Wang et al., 2024b). SWE-agent, OpenHands, Agentless, AutoCodeRover, CodePlan, and SWE-Search instantiate repository agents with different interfaces, planning, analysis, and search strategies (Yang et al., 2024; Wang et al., 2025; Xia et al., 2024; Zhang et al., 2024; Bairi et al., 2024; Antoniades et al., 2025).

Training coding agents from verified trajectories. Agent-tuning and self-training methods construct interaction traces, generate candidates, retain outputs supported by outcome signals, and optimize on the selected data (Chen et al., 2023; Zeng et al., 2024; Chen et al., 2024; Zelikman et al., 2022; Gulcehre et al., 2023; Singh et al., 2024; Yuan et al., 2023). SWE-Gym, SWE-smith, and R2E-Gym scale this recipe for software engineering (Pan et al., 2025; Yang et al., 2025b; Jain et al., 2025). Agent-RLVR uses environment rewards, and FailForge studies gold-patch-informed guidance (Da et al., 2025; Lv et al., 2026). GS-SFT compares when answer-derived guidance enters coding-agent trajectory collection and how those trajectories are represented for SFT.

Privileged and process supervision. Learning using privileged information and distillation separates training-only teacher signals from deployment inputs (Vapnik & Vashist, 2009; Lopez-Paz et al., 2016). Privileged evidence can shape supervised targets even when it is absent from the student input (Chen et al., 2020; Hase et al., 2020; Geirhos et al., 2020; Wang et al., 2023). Process supervision evaluates intermediate reasoning (Lightman et al., 2024; Wang et al., 2024a). GS-SFT uses reference solutions to generate trajectories, filters them with executable tests, and evaluates the trained agent without task-specific guidance.

3 Problem Setup and Guided Rollout Formalization

Repository tasks and ReAct trajectories. A repository task is

$$x = (q, \mathcal{R}_0, \mathcal{E}, V),$$

where q is the issue description, $\mathcal{R}_0$ is the initial repository, $\mathcal{E}$ is the tool environment, and V is the terminal verifier. The agent interacts in a ReAct loop. Let h_t denote the interaction history up to step t:

$$h_t = (c_0;\, r_1, a_1, o_1, \ldots, r_{t-1}, a_{t-1}, o_{t-1}).$$

Here c_0 is the initial conditioning context. At step t, the policy samples $(r_t, a_t) \sim \pi_\theta(\cdot \mid h_t)$; the environment executes a_t, returns observation o_t, and may update the repository from $\mathcal{R}_{t-1}$ to $\mathcal{R}_t$. A completed rollout is the interaction record

$$\tau = (c_0;\, r_1, a_1, o_1, \ldots, r_T, a_T, o_T).$$

The resulting patch is extracted after the rollout as the repository difference

$$p(\tau) = \mathrm{diff}(\mathcal{R}_0, \mathcal{R}_T).$$

The verifier evaluates this final repository change, giving $V_x(\tau) \in \{0, 1\}$.

Gold patch and reference-conditioned guidance. Trajectory collection may use a privileged *gold patch* $p^\star(x)$: the developer / reference code modification that solves x. Let ℓ index a guidance subtype; in our experiments, it distinguishes different guidance information strengths within a timing family. We write G for the actual guidance message. Pre guidance is generated before interaction:

$$G_\ell^{\mathrm{pre}}(x) = g_\ell^{\mathrm{pre}}\big(p^\star(x), q, \mathcal{R}_0\big). \tag{1}$$

Guidance and q are peer textual inputs to the coding agent during trajectory collection. L6 uses oracle reference-patch framing.

Timing families. Without task-specific guidance, the initial context is $c_0 = (q, \mathcal{R}_0)$. Pre guidance places the message beside the problem statement from the first step,

$$c_0^{\mathrm{pre},\ell} = \big(q, G_\ell^{\mathrm{pre}}(x), \mathcal{R}_0\big), \qquad \tau_{\mathrm{pre},\ell} \sim \pi_\theta\big(\cdot \mid c_0^{\mathrm{pre},\ell}\big).$$

Post guidance first runs an unguided attempt $\tau^{(0)}$. After failure, the teacher may use that attempt privately to translate failure evidence into retry-oriented guidance:

$$G_\ell^{\mathrm{post}}(x; \tau^{(0)}) = g_\ell^{\mathrm{post}}\big(p^\star(x), q, \mathcal{R}_0, \tau^{(0)}\big), \quad V_x(\tau^{(0)}) = 0.$$

The fresh retry receives the translated message as a peer to q, but not the raw failed trajectory:

$$c_0^{\mathrm{post},\ell} = \big(q, G_\ell^{\mathrm{post}}(x; \tau^{(0)}), \mathcal{R}_0\big), \qquad \tau_{\mathrm{post},\ell} \sim \pi_\theta\big(\cdot \mid c_0^{\mathrm{post},\ell}\big).$$

Intra guidance keeps $c_0 = (q, \mathcal{R}_0)$ but, at an eligible triggered step, produces step-local guidance and may rewrite the proposed response:

$$G_{\ell,t} = g_\ell\big(p^\star(x), q, \mathcal{R}_0, h_t\big), \qquad (\widetilde{r}_t, \widetilde{a}_t) = \rho(r_t, a_t, G_{\ell,t}).$$

The environment executes $\widetilde{a}_t$, so rewritten actions change later observations. In the reported historical Intra construction, the guidance message is not independently serialized as SFT input, but teacher-shaped reasoning or actions can remain in the target.

Training data and evaluation. Let $\mathcal{D}_0$ denote the verifier-accepted trajectories collected without task-specific guidance, and let $\mathcal{D}_\ell^G$ denote the additional verifier-accepted trajectories collected with guidance subtype ℓ. After standard SFT formatting and filtering, the training data are simply their multiset union:

$$\mathcal{D}_\ell = \mathcal{D}_0 \uplus \mathcal{D}_\ell^G. \tag{2}$$

The base language model is then trained with ordinary SFT on $\mathcal{D}_\ell$:

$$\theta_\ell = \mathrm{SFT}(\theta_{\mathrm{base}}, \mathcal{D}_\ell).$$

Table 1: Six guidance strengths used for Pre, Post, and Intra trajectory collection.

Level	Public role	Information provided	Information withheld
L1	Navigation only	Relevant file paths or repository areas; Post gives retry-oriented file navigation	Symbols, implementation, and patch text
L2	File and object navigation	Relevant files plus target symbols, classes, or functions	Behavioral specification and implementation
L3	Directional behavior specification	Required behavior, observed mismatch, and behavioral constraints	Ordered edit plan and patch-adjacent details
L4	Directional implementation / retry plan	Ordered repair direction, dependencies, and validation plan	Patch-adjacent solution blueprint and verbatim code
L5	Patch-adjacent solution / retry blueprint	Concrete edit blueprint derived from the reference solution	Verbatim reference-patch insertion
L6	Oracle reference patch	Pre uses deterministic patch framing; Post uses notes plus patch with fallback	No abstraction constraint applies

For the executable task set $\mathcal{X}_{\text{exec}}$, Total Supply and Added Supply are

$$S_\ell^{\text{total}} = \frac{|\mathcal{D}_\ell|}{|\mathcal{X}_{\text{exec}}|} \times 100\%, \qquad S_\ell^{\text{add}} = \frac{|\mathcal{D}_\ell^G|}{|\mathcal{X}_{\text{exec}}|} \times 100\%.$$

These quantities are reported as Supply percentages.

The trained model θ_ℓ is placed back into the coding-agent scaffold and evaluated without task-specific guidance. **Utility is the task resolved rate on the test set.** Thus Supply measures how much training data a guidance subtype produces, whereas Utility measures how well the resulting coding agent solves held-out software-engineering tasks.

4 Guided Trajectory Data Construction

We distinguish *executable task data* from *trajectory data*. An executable software-engineering task provides a problem statement, repository, runnable environment, and tests; it exists as a reusable data asset before any agent is run. A trajectory is produced only by running a coding agent on that task. If repeated rollouts never pass the tests, the executable task produces no retained trajectory for SFT.

GS-SFT uses answer-derived guidance to convert more executable tasks into successful trajectory data. We apply guidance only to *hard tasks*: tasks for which the initial no-guidance rollouts produce no test-passing trajectory. Guidance enters before, after, or during interaction, and the same tests verify the resulting repository change. The additional successful trajectories are combined with initially successful no-guidance trajectories for ordinary SFT. We call this process *trajectory collection*; Supply is the retained-example count normalized by the executable task pool and reported as a percentage.

Figure 1 summarizes the guidance strengths, rollout timing, and training pipeline.

4.1 Guidance source and six strengths

Guidance is derived from the gold patch $p^\star(x)$ together with the problem statement and repository context. The gold patch is used only during trajectory collection; evaluation provides no task-specific guidance. For Post, the generator may additionally use an unsuccessful attempt to produce a retry-oriented hint.

We instantiate six guidance strengths. L1 provides file or repository navigation; L2 adds target symbols or objects; L3 specifies the desired behavior; L4 gives an implementation or retry plan; and L5 provides a patch-adjacent solution blueprint without inserting the reference patch verbatim. L6 directly exposes the reference patch and is treated as an oracle rather than an ordinary strength.

Table 1 details the information provided at each level.

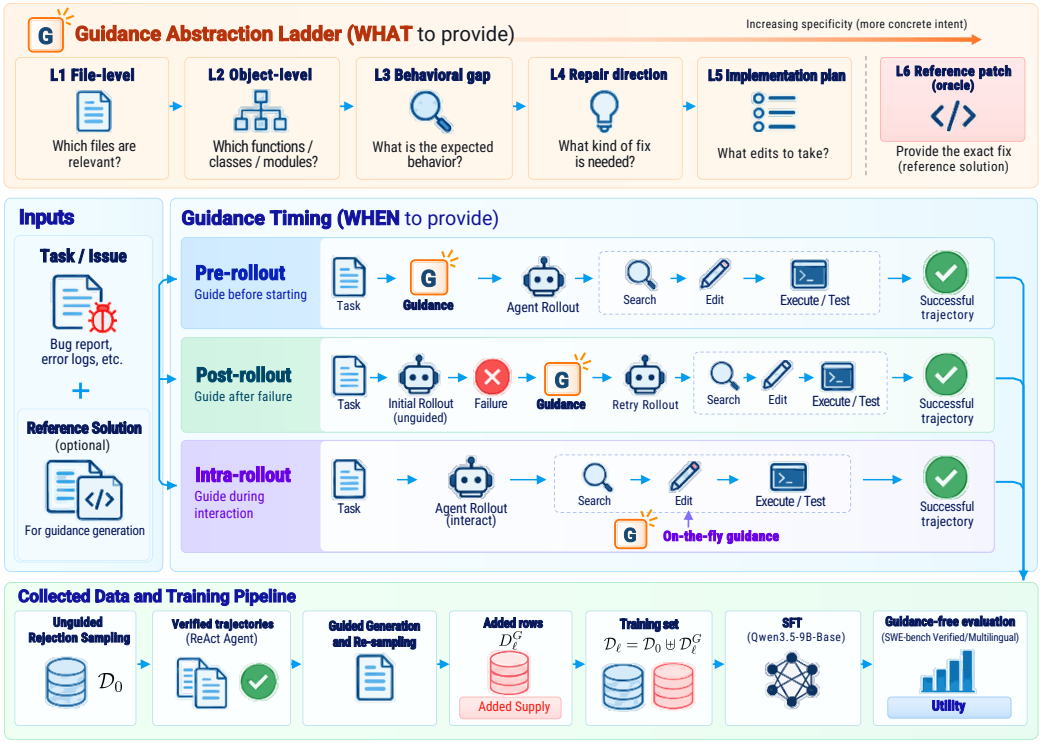


Figure 1: GS-SFT overview. The top panel shows the six guidance strengths, from file navigation to the reference-patch oracle. The middle panel shows when Pre, Post, and Intra guidance enters an agent rollout. The bottom panel shows that verified guided trajectories from hard tasks augment no-guidance rejection-sampling data before SFT, followed by guidance-free evaluation.

4.2 Three Guided Rollout Families

Pre guidance. Pre guidance is generated directly from the answer information and given to the agent before its first action. It can therefore guide repository navigation, reasoning, editing, and validation throughout the rollout.

Post guidance. The agent first performs a no-guidance rollout. If it fails, the generator combines the failed attempt with the answer information to produce a retry hint. A fresh agent then starts a new rollout using that hint.

Intra guidance. Guidance is generated during the rollout at state-changing steps. It uses the current interaction history and answer information to refine the agent's reasoning or next action before execution continues.

4.3 Guided-Success Augmentation

Each family is used to sample additional trajectories on the hard-task subset under the same executable verifier. A trajectory is retained when the final repository change passes the task tests. For guidance subtype ℓ, these additional successful trajectories form $\mathcal{D}_\ell^G$ and are combined with the initially successful no-guidance trajectories $\mathcal{D}_0$:

$$\mathcal{D}_\ell = \mathcal{D}_0 \uplus \mathcal{D}_\ell^G.$$

The resulting dataset is used for ordinary SFT. At evaluation time, the trained coding agent receives only the test issue and repository, with no task-specific guidance. Experimental preparation and

Table 2: Composition of the 4,000 executable software-engineering tasks used for trajectory collection.

Dataset source

Source	Tasks	Share
SWE-ReBench	2,102	52.55%
Multi-SWE-RL	1,561	39.02%
SWE-Gym	213	5.33%
SWE-bench Extra	124	3.10%
Total	4,000	100.00%

Language (top five)

Language	Tasks	Share
Python	3,412	85.30%
TypeScript	111	2.77%
Java	107	2.67%
JavaScript	98	2.45%
Go	91	2.27%
Other (8 languages)	181	4.53%
Total	4,000	100.00%

visibility controls are described in Section 5; filtering, length handling, and loss masking are given in Appendix A.

5 Experimental Setup

5.1 Executable task pool and baseline trajectory collection

We collect trajectories from 4,000 executable software-engineering tasks in the SWE-bench setting (Jimenez et al., 2024). Each task provides an issue, repository snapshot, runnable environment, and tests. The coding agent can inspect and modify the repository and run commands; after interaction, the evaluated patch is extracted from the difference between the initial and final repository states. Table 2 summarizes the task sources and languages.

The rollout policy is **Seed 2.0 Pro**, served through Volcano Engine Ark.[1] We first sample three no-guidance rollouts for every task, yielding 12,000 candidate trajectories. A task is counted as solved when at least one rollout passes its tests. This baseline solves 1,522 of 4,000 tasks, giving Pass@3 of 38.05%; the remaining 2,478 tasks form the *hard-task pool*. Standard SFT preparation retains 1,480 no-guidance examples from the successful trajectories. Guided trajectory collection targets the hard-task pool.

5.2 Guided trajectory collection and SFT

For each hard task, we apply the Pre, Post, or Intra family at one of six guidance strengths and retain any new trajectory that passes the same tests. Each condition adds these guidance-produced examples to the common 1,480-example no-guidance set, as defined in Equation 2. Table 5 reports the added examples together with Supply and Utility.

The primary experiment therefore contains one no-guidance baseline and L1 to L6 for each of the three guidance families. L1 to L5 are ordinary guidance strengths; L6 directly exposes the reference patch and is an oracle. All resulting datasets are used to fine-tune `Qwen3.5-9B-Base` with ordinary SFT. Historical Pre/Post conditions use two epochs, with the endpoint step determined by training-set size.

5.3 Trajectory preparation and visibility controls

Verifier-accepted trajectories are converted into the model's conversation format before SFT. The exact completion, length, and target-masking rules are reported in Appendix A. Pre and Post retain their hint as part of the SFT input. Historical Intra retains the guided reasoning and actions but not a separate guidance message.

The current Intra visibility controls compare two preparations of the same guided trajectory source: one includes a cleaned guidance reminder, while the other omits that reminder. Both use the same

[1] `https://ark.volcengine.com/region:cn-beijing/model/detail?Id=doubao-seed-2-0-pro`

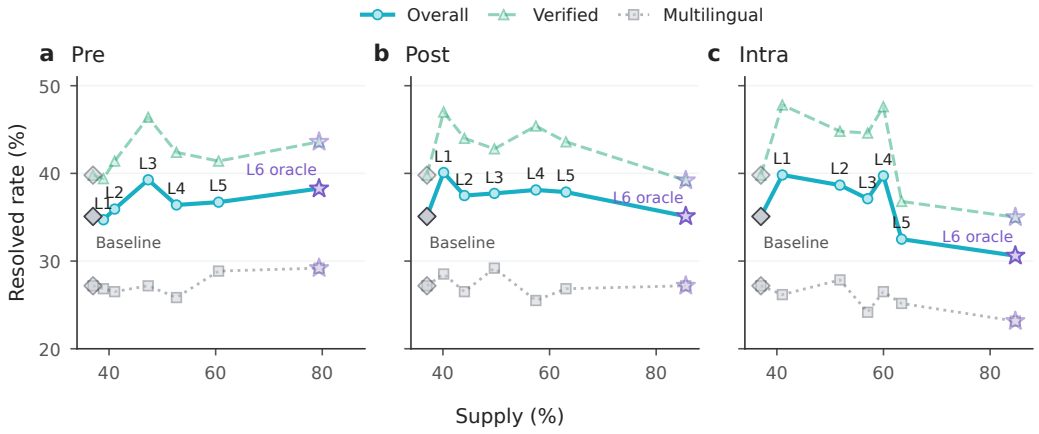


Figure 2: Supply and Utility across Pre, Post, and Intra guidance. Curves connect the shared baseline through L1 to L6. Appendix Table 5 reports the added examples and complete values.

base-model designation, optimizer schedule, step budget, and evaluation requests. The released comparison covers L1 to L4.

5.4 Evaluation

Every trained policy is evaluated without task-specific guidance on 798 tasks: 500 SWE-bench Verified tasks (Jimenez et al., 2024) and 298 SWE-bench Multilingual tasks (Yang et al., 2025b). Utility is the proportion of tasks resolved by the coding agent. The shared no-guidance baseline resolves $280/798 = 35.09\%$; segment results use fixed denominators of 500 and 298. Context-window, interaction-budget, and output-length exhaustion count as agent failures, while registered infrastructure failures are excluded.

6 Results

Guidance addresses the collection bottleneck, but Supply and Utility diverge. Figure 2 shows the family-level trends, while Appendix Table 5 reports the added examples, Supply, and Utility for every condition. The complete resolved-rate breakdown by Overall, Verified, and Multilingual subsets appears in Appendix Table 6.

6.1 Guidance recovers training data from hard tasks

Unguided rejection sampling wastes most executable tasks. The three no-guidance rollouts solve only 1,522 of the 4,000 tasks, corresponding to Pass@3 of 38.05%. Thus 2,478 tasks with valid environments and tests produce no successful baseline trajectory. After preparation, the no-guidance set contains 1,480 SFT examples and has Supply of 37.00%. This gap between executable tasks and retained trajectories is the data-utilization opportunity targeted by guided collection.

Guidance recovers data in the expected order. From L1 to L5, every family produces monotonically more added examples. Pre grows from 78 to 942 examples, raising Supply from 38.95% to 60.55%; Post grows from 125 to 1,043, raising Supply from 40.13% to 63.08%; and Intra grows from 161 to 1,056, raising Supply from 41.03% to 63.40%. At every matched level, Post produces more examples than Pre because it can use information from an initial failed rollout. L6 yields the largest Supply: 79.40%, 85.55%, and 84.73% for Pre, Post, and Intra. The collection result is therefore consistent across all three intervention timings: stronger answer-derived hints make hard tasks more likely to yield usable SFT data.

Table 3: Utility (%) under guidance-text removal or reinsertion. For offline Pre/Post trajectories, Keep retains the hint that conditioned the rollout and Remove strips it from the SFT input. Historical Keep/Remove evaluations are cross-batch. Intra trajectories omit a standalone guidance message by default; Reinsert adds cleaned guidance to the preceding turn as a system reminder.

Level	Pre (offline)		Post (offline)		Intra (online)	
	Keep	Remove	Keep	Remove	Removed (default)	Reinsert reminder
L1	34.71	32.83	40.10	33.33	36.76	38.90
L2	35.93	32.71	37.47	31.20	40.15	44.54
L3	39.27	35.63	37.72	32.58	36.72	36.97
L4	36.40	31.20	38.11	33.08	41.85	38.47
L5	36.72	31.95	37.87	32.08		

6.2 Utility peaks before the strongest guidance

More Supply does not produce more Utility. Figure 2 visualizes this divergence. Each family reaches its best ordinary Utility before L5 and declines under stronger guidance. Pre rises from 34.71% at L1 to 39.27% at L3, then falls to 36.72% at L5. Post peaks at L1 with 40.10% and ends at 37.87% at L5. Intra reaches 39.82% at L1 and 39.72% at L4, but drops to 32.50% at L5. Across all three families, moderately informative guidance outperforms the strongest ordinary guidance.

Post's collection advantage does not translate into a consistent Utility advantage. Excluding L1, the Utility difference between matched Post and Pre conditions ranges only from -1.55 to $+1.71$ points. Intra likewise produces substantial additional Supply without dominating the offline families.

The full answer is not the best supervision. Pre L6 reaches 38.27%, below the Pre L3 peak despite much larger Supply. Post L6 falls to 35.11%, the lowest Post result, and Intra L6 falls to 30.61%, the lowest result in the sweep. Directly exposing the answer maximizes trajectory collection but can degrade downstream resolution; effective SFT data construction should use answer information selectively.

7 Discussion

7.1 Offline guidance should remain in the SFT input

Pre and Post trajectories are generated while the agent can read the hint. This raises a practical question for SFT construction: after a successful trajectory is collected, should the hint remain in the training input or be removed? Removing it leaves the same actions and reasoning but hides information that originally conditioned those decisions.

We test this choice by training on the same successful trajectories with the Pre/Post hint either kept or removed. Keeping the hint yields higher observed Utility in all ten pairs. In the source-defined seven-pair subset, which contains Post L1 to L5 and Pre L4 to L5, the advantage ranges from $+4.76$ to $+6.77$ percentage points, with a mean of $+5.60$ points. For offline guidance, the information used to generate a trajectory should remain visible when that trajectory becomes SFT data.

7.2 Online Intra guidance behaves differently

Intra guidance is consumed online while the trajectory is being generated. By default, the guidance message itself is removed before SFT, while the refined reasoning and actions remain. We ask whether explicitly restoring that text helps the model understand the resulting trajectory.

For the restored variant, we clean the step-level guidance and insert it into the preceding turn as a system reminder. The removed and reinserted variants use the same evaluation requests. As shown in Table 3, reinsertion improves L1 and L2, changes little at L3, and reduces Utility at L4. Unlike offline trajectories, online guidance has already shaped the supervised response; adding the text again is therefore not uniformly beneficial. The conclusion is specific: keep guidance for offline Pre/Post SFT, but treat explicit Intra reminder insertion as a level-dependent design choice.

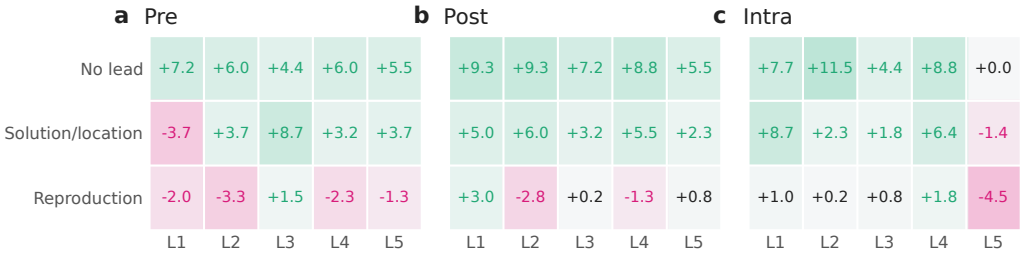


Figure 3: Utility difference from the task-type-specific baseline in percentage points. Green cells improve over baseline, rose cells decline, and gray cells are near zero. Every ordinary condition from L1 to L5 is shown.

7.3 Where the Utility gains occur

We partition the 798 evaluation cases by information already present in the issue: 182 contain neither a solution/location clue nor reproduction evidence ("no lead"), 218 contain a solution or location clue, and 398 contain reproduction steps or a traceback. Figure 3 reports each ordinary condition from L1 to L5 relative to the corresponding task-type baseline.

No-lead cases improve in nearly every ordinary condition, with gains of up to $+11.5$ points. Solution/location cases are also often improved, whereas reproduction cases are mixed and usually close to baseline. This shows why Overall Utility is too coarse: guidance primarily helps where the issue provides little direct direction and remains useful, though less consistently, when a likely solution or location is already named.

Together, the visibility and task-type results refine the main relationship between Supply and Utility: useful guided data must preserve the information that conditioned offline actions, and its benefit is largest when the issue itself provides little direction.

8 Conclusion

GS-SFT uses answer-derived Pre, Post, and Intra guidance to recover successful trajectories from executable tasks that no-guidance rollouts fail to solve. Guidance consistently increases Supply, but Utility peaks before the strongest conditions; direct reference patches produce the most data without producing the strongest models. Moderately informative guidance is therefore more useful than blindly exposing the full answer, and offline hints should remain in the SFT input that accompanies the trajectories they conditioned.

This result may inform privileged-information methods such as on-policy self-distillation (OPSD) (Zhao et al., 2026), where the form and consumption of privileged information may matter more than its completeness. Future work should test whether similar trends hold when privileged information is introduced during reinforcement learning or other on-policy post-training stages.

References

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Yang Wang. SWE-Search: Enhancing software agents with monte carlo tree search and iterative refinement. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a1e6783e4d739196cad3336f12d402bf-Abstract-Conference.html.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025. doi: 10.52202/085713-0788.

URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/21bec6ace947b1b58967b945c8ac0f10-Abstract-Datasets_and_Benchmarks_Track.html.

Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D. C., Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B. Ashok, and Shashank Shet. CodePlan: Repository-level coding using LLMs and planning. *Proceedings of the ACM on Software Engineering*, 1(FSE):675–698, 2024. doi: 10.1145/3643757. URL https://doi.org/10.1145/3643757.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. FireAct: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023. URL https://arxiv.org/abs/2310.05915.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 66–75, 2020. URL https://proceedings.mlr.press/v100/chen20a.html.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 9354–9366, 2024. URL https://aclanthology.org/2024.findings-acl.557/.

Jeff Da, Clinton Wang, Xiang Deng, Yuntao Ma, Nikhil Barhate, and Sean Hendryx. Agent-RLVR: Training software engineering agents via guidance and environment rewards. *arXiv preprint arXiv:2506.11425*, 2025. URL https://arxiv.org/abs/2506.11425v2. Version 2 inspected.

Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. CrossCodeEval: A diverse and multilingual benchmark for cross-file code completion. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 36, 2023. URL https://openreview.net/forum?id=wgDcbBMSfh.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2:665–673, 2020. doi: 10.1038/s42256-020-00257-z. URL https://arxiv.org/abs/2004.07780.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (ReST) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023. URL https://arxiv.org/abs/2308.08998.

Peter Hase, Shiyue Zhang, Harry Xie, and Mohit Bansal. Leakage-adjusted simulatability: Can models generate non-trivial explanations of their behavior in natural language? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4351–4367, 2020. doi: 10.18653/v1/2020.findings-emnlp.390. URL https://aclanthology.org/2020.findings-emnlp.390/.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 34, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract-round2.html.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environments and hybrid verifiers for scaling open-weights software engineering agents. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2504.07164.

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*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Tianyang Liu, Canwen Xu, and Julian McAuley. RepoBench: Benchmarking repository-level code auto-completion systems. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/d191ba4c8923ed8fd8935b7c98658b5f-Abstract-Conference.html.

David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1511.03643.

Dongyi Lv, Fushun E, Aichen Cai, Liang Huang, Ya Zhang, Qiuyu Ding, Canhui Wu, Zhi Wang, Yuesong Zhang, Jiaqi Wang, and Nan Duan. FailForge: Distilling procedural competence from persistent failures into code agents. *arXiv preprint arXiv:2608.08570*, 2026. URL https://arxiv.org/abs/2608.08570v1. Version 1 inspected.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. SWE-Lancer: Can frontier LLMs earn $1 million from real-world freelance software engineering? In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 44412–44450, 2025. URL https://proceedings.mlr.press/v267/miserendino25a.html.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47717–47737, 2025. URL https://proceedings.mlr.press/v267/pan25g.html.

Muhammad Shihab Rashid, Christian Bock, Yuan Zhuang, Alexander Buchholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, Anoop Deoras, Giovanni Zappella, and Laurent Callot. SWE-PolyBench: A multi-language benchmark for repository-level evaluation of coding agents. *arXiv preprint arXiv:2504.08703*, 2025. URL https://arxiv.org/abs/2504.08703.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=lNAyUngGFK.

Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5–6):544–557, 2009. doi: 10.1016/j.neunet.2009.06.042. URL https://doi.org/10.1016/j.neunet.2009.06.042.

Peifeng Wang, Zhengyang Wang, Zheng Li, Yifan Gao, Bing Yin, and Xiang Ren. SCOTT: Self-consistent chain-of-thought distillation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://aclanthology.org/2023.acl-long.304/.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 9426–9439, 2024a. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232, 2024b. URL https://proceedings.mlr.press/v235/wang24h.html.

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*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a4b6ad6b48850c0c331d1259fc66a69c-Abstract-Conference.html.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024. URL https://arxiv.org/abs/2407.01489.

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*, 2024. URL https://arxiv.org/abs/2405.15793.

John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida I. Wang, and Ofir Press. SWE-bench multimodal: Do AI systems generalize to visual software domains? In *International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=riTiq3i21b.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023. URL https://arxiv.org/abs/2308.01825.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Kai Shen, and Liang Xiang. Multi-SWE-bench: A multilingual benchmark for issue resolving. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2504.02605.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. URL https://aclanthology.org/2024.findings-acl.181/.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. RepoCoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2471–2484, 2023. doi: 10.18653/v1/2023.emnlp-main.151. URL https://aclanthology.org/2023.emnlp-main.151/.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. SWE-bench goes live! In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2505.23419.

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*, pp. 1592–1604, 2024. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026. URL https://arxiv.org/abs/2601.18734.

A Experimental Settings

A.1 Executable task data

The trajectory-collection pool contains 4,000 executable software-engineering tasks from SWE-ReBench, Multi-SWE-RL, SWE-Gym, and SWE-bench Extra. Each task includes an issue, repository snapshot, runnable environment, and tests. Table 4 reports the complete 13-language distribution.

Table 4: Language distribution of the 4,000 executable software-engineering tasks.

Language	Task instances	Share
Python	3,412	85.30%
TypeScript	111	2.77%
Java	107	2.67%
JavaScript	98	2.45%
Go	91	2.27%
Ruby	60	1.50%
C	26	0.65%
Kotlin	24	0.60%
Rust	18	0.45%
C++	17	0.43%
PHP	14	0.35%
Scala	13	0.33%
Swift	9	0.22%
Total	4,000	100.00%

We use Seed 2.0 Pro as the rollout policy and sample three no-guidance trajectories for each task. A task is a hard task when none of these trajectories passes the tests. Guidance is applied only to this hard-task pool.

Table 5: Guidance-produced examples, Supply, and Utility. Added is the number of successful guidance-produced SFT examples; Supply normalizes total retained examples by the 4,000 executable tasks; Utility is task resolved rate after SFT.

	Pre-rollout			Post-rollout			Intra-rollout		
Level	Added	Supply (%)	Utility (%)	Added	Supply (%)	Utility (%)	Added	Supply (%)	Utility (%)
Shared baseline	Added = 0; Supply = 37.00%; Utility = 35.09% (280/798)								
L1	78	38.95%	34.71%	125	40.13%	40.10%	161	41.03%	39.82%
L2	162	41.05%	35.93%	281	44.03%	37.47%	593	51.83%	38.66%
L3	414	47.35%	39.27%	506	49.65%	37.72%	801	57.03%	37.11%
L4	626	52.65%	36.40%	817	57.43%	38.11%	919	59.98%	39.72%
L5	942	60.55%	36.72%	1,043	63.08%	37.87%	1,056	63.40%	32.50%
Direct-reference-patch oracle (excluded from ordinary ladder)									
L6 (Oracle)†	1,696	79.40%	38.27%	1,942	85.55%	35.11%	1,909	84.73%	30.61%

A.2 Coding-agent interface

The rollout model operates through a minimal ReAct repository agent that alternates reasoning, tool execution, and environment observation. It exposes exactly two tools: `bash` executes Linux commands for repository inspection and testing, while `str_replace` performs targeted file edits. The interaction history records reasoning, tool calls, and observations; the final patch is extracted from the difference between the initial and final repository states.

A.3 SFT configuration

Verifier-accepted trajectories are converted into the model's conversation format, restricted to completed nonempty interactions, filtered to a maximum length of 131,072 tokens, and assigned assistant-token loss masks. We fine-tune Qwen3.5-9B-Base with full-parameter SFT using `ms-swift`. The common training configuration uses two epochs, learning rate 5×10^{-6}, a 131,072-token sequence length, and ZeRO-2 distributed optimization. The number of optimization steps varies with the number of retained examples.

A.4 Evaluation

We evaluate the fine-tuned model in the same minimal coding-agent scaffold without task-specific guidance. The evaluation set contains 798 tasks: 500 SWE-bench Verified and 298 SWE-bench Multilingual. A task is resolved when the final repository change passes the benchmark tests. The nominal evaluation configuration uses temperature 0.9, top-p 0.95, a maximum output length of 8,192 tokens, at most 200 interaction turns, and a 138,240-token context window. Utility is the resolved rate over this combined evaluation set.

B Experimental Results

B.1 Supply and Utility by condition

Table 5 reports added examples, Supply, and Utility for every guidance family and strength.

B.2 Complete resolved-rate results

Table 6 reports Overall, Verified, and Multilingual resolved rates for all 19 conditions.

B.3 Issue-information categories

For the task-type analysis in Section 7.3, we classify each evaluation issue according to the information available before agent interaction:

- **No lead** ($n = 182$): the issue contains neither an explicit solution/location clue nor reproduction evidence.

Table 6: Complete resolved-rate results for all 19 conditions. Overall uses the nominal 798-task endpoint; Verified and Multilingual use fixed denominators of 500 and 298.

Condition	Overall Utility (%)	Verified (%)	Multilingual (%)
Shared baseline	35.09%	39.80%	27.18%
Pre L1	34.71%	39.40%	26.85%
Pre L2	35.93%	41.40%	26.51%
Pre L3	39.27%	46.40%	27.18%
Pre L4	36.40%	42.40%	25.84%
Pre L5	36.72%	41.40%	28.86%
Pre L6	38.27%	43.60%	29.19%
Post L1	40.10%	47.00%	28.52%
Post L2	37.47%	44.00%	26.51%
Post L3	37.72%	42.80%	29.19%
Post L4	38.11%	45.40%	25.50%
Post L5	37.87%	43.60%	26.85%
Post L6	35.11%	39.20%	27.18%
Intra L1	39.82%	47.80%	26.17%
Intra L2	38.66%	44.80%	27.85%
Intra L3	37.11%	44.60%	24.16%
Intra L4	39.72%	47.60%	26.51%
Intra L5	32.50%	36.80%	25.17%
Intra L6	30.61%	35.00%	23.15%

- **Solution/location** ($n = 218$): the issue identifies a likely solution, relevant file, or code location.
- **Reproduction** ($n = 398$): the issue contains reproduction steps, an error trace, or equivalent failure evidence but no solution/location clue.

The categories are mutually exclusive; solution/location takes precedence when both types of information appear. Figure 3 reports every ordinary Pre, Post, and Intra result across these three categories.

C Guidance Implementation

All three methods derive guidance from the problem statement q, initial repository $\mathcal{R}_0$, and gold patch $p^\star(x)$. The guidance subtype ℓ selects one of the conditions from L1 to L6 and determines how much answer information is exposed. The methods differ only in when the guidance is produced and how it is used to generate a new trajectory.

C.1 Pre guidance

Pre guidance is produced before the agent begins interacting with the repository. The generated hint is placed beside the problem statement and remains available throughout the rollout.

```
G = generate_pre_hint(problem, repository, gold_patch, level)
trajectory = run_agent(problem, repository, guidance=G)
if tests_pass(diff(initial_repository, final_repository)):
    retain(trajectory)
```

C.2 Post guidance

Post guidance uses an unsuccessful no-guidance attempt to produce a retry-oriented hint. A fresh agent then starts a new rollout from the initial repository using the translated hint.

```
failed = run_agent(problem, repository)
G = generate_retry_hint(problem, repository, gold_patch, failed, level)
```

```
trajectory = run_fresh_agent(problem, repository, guidance=G)
if tests_pass(diff(initial_repository, final_repository)):
    retain(trajectory)
```

C.3 INTRA GUIDANCE

Intra guidance is produced during the rollout at state-changing steps. The generator conditions on the current interaction history and gold patch, then refines the proposed reasoning or action before the environment executes it.

```
history = initialize(problem, repository)
while not finished:
    response = agent(history)
    if response changes repository state:
        G = generate_step_hint(problem, repository, gold_patch, history, level)
        response = refine(response, G)
    observation = execute(response)
    history.append(response, observation)
if tests_pass(diff(initial_repository, final_repository)):
    retain(history)
```

For every family and guidance subtype, successful guided trajectories are added to the retained no-guidance trajectories and used for ordinary SFT.

D ETHICS STATEMENT

The executable tasks used in this study are derived from publicly available open-source repositories and issue histories. Nevertheless, source-code licenses and repository-specific usage conditions continue to apply, particularly when reference patches are used as privileged information. Researchers should preserve provenance, respect applicable licenses, and avoid redistributing code or trajectories that contain restricted repository content.

Coding agents can generate insecure, vulnerable, or malicious changes even when those changes pass an incomplete test suite. Guidance generated from a reference solution can also reproduce undesirable behavior present in that solution. Rollout collection, evaluation, and deployment should therefore occur in isolated containers with restricted filesystem and network access. Generated patches should undergo human review, additional testing, and security analysis before use in production systems.

E REPRODUCIBILITY STATEMENT

Appendix A reports the task composition, rollout model, agent interface, SFT hyperparameters, and evaluation configuration. Appendix C specifies the three guidance production procedures using pseudocode.

The supplementary material provides redacted extracts of the mainline guidance code as a basis for reproduction. It includes the Pre, Post, and Intra prompts for L1 to L6, task and trajectory data contracts, hard task selection, trajectory verification, dataset assembly, aggregate results, and deterministic table and figure generators.

F USE OF LLMS

Large language models are central experimental components of this work: Seed 2.0 Pro produces coding-agent rollouts, answer-derived guidance is generated by an LLM during trajectory collection, and Qwen3.5-9B-Base is fine-tuned on the resulting data. LLM-based tools were also used for code completion and language refinement during development of the research artifacts and manuscript. The research questions, experimental design, analysis decisions, and final claims were determined and verified by the authors, who take responsibility for the content and conclusions of the paper.