TERMINALEVO: EVOLVING ENVIRONMENTS AT THE CAPABILITY FRONTIER OF TERMINAL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training terminal agents requires executable environments that provide useful experience for the target agent. However, increasing task complexity alone does not ensure that the resulting supervision matches the agent's capabilities. We introduce **TerminalEvo**, a framework that evolves terminal environments toward a target agent's empirical capability frontier using complementary outcome and behavioral feedback. Repeated rollouts from a fixed agent determine whether a task should continue evolving, while action–observation trajectories guide which dependencies or constraints to introduce. A large language model editor translates this analysis into a transformation contract, which guides solution-first task construction and execution-based validation. A stronger teacher agent then generates successful demonstrations on the resulting tasks, yielding a separate training corpus for each student. We evaluate Qwen3.5-4B, 9B, and 27B on Long-Horizon Terminal-Bench, Terminal-Bench 2.1, and SWE-bench Verified under approximately matched supervised fine-tuning budgets measured in loss-bearing tokens. Compared with four agent-training datasets, TerminalEvo achieves the best performance in seven of nine model–benchmark settings and ranks second in the remaining two. Direct preference optimization (DPO) after SFT further improves performance in all nine settings. These results demonstrate the effectiveness of student-conditioned environment evolution for generating terminal-agent training supervision.

1 INTRODUCTION

Language-model agents are increasingly being deployed in resource-constrained settings, including personal devices, edge systems, enterprise environments, and large-scale services where many agents operate concurrently (Erdogan et al., 2024; Shlomov et al., 2026). In these scenarios, compact language models offer advantages in inference cost, latency, privacy, and deployment flexibility, making them an important foundation for practical AI agents (Liu et al., 2024b; Peng et al., 2024; Erdogan et al., 2024). Their limited capacity, however, makes training particularly sensitive to the difficulty of the available experience (Tang & Zhao, 2026). More data or more complex tasks do not necessarily lead to better learning (Zhou et al., 2023; Hase et al., 2024): when a task demands capabilities far beyond those of the student, even a valid demonstration may provide limited training value (Li et al., 2025; Tang & Zhao, 2026). This raises a fundamental question for small agents: *what interaction experience is actually learnable and useful for a model with limited capacity?*

This question becomes especially important when agents learn through executable environments, where task difficulty depends not only on the instruction but also on the observations, actions, state changes, and dependencies encountered during execution. AgentBench (Liu et al., 2024a) studies language-model agents across diverse interactive environments, while Terminal-Bench (Merrill et al., 2026b) provides a representative terminal-agent setting in which agents operate through command-line interfaces over stateful environments to complete realistic multi-step tasks. Recent environment-generation methods such as SETA (Shen et al., 2026) further highlight the need for executable tasks with reliable verification. Effective data generation in this setting therefore requires environments whose demands are appropriate for the target student.

Recent progress in synthetic data generation provides a scalable basis for this goal. Self-Instruct (Wang et al., 2023) bootstraps instruction corpora from a small set of human-written seeds,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

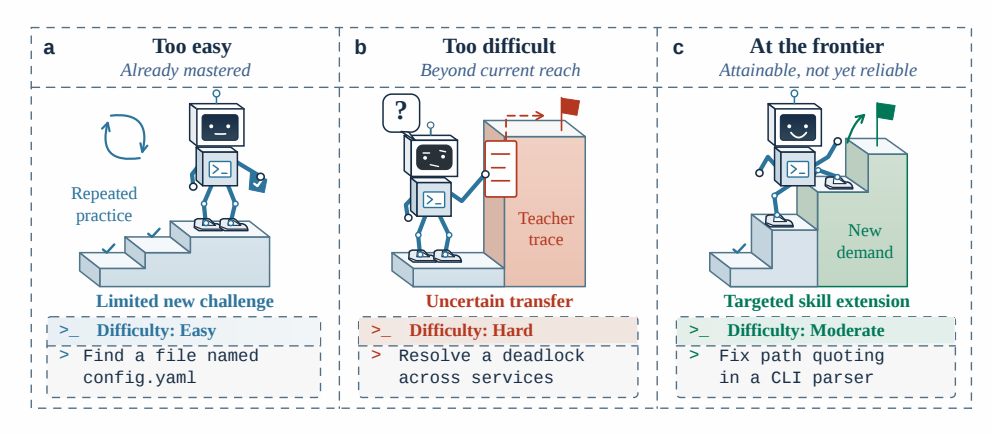

Figure 1: **The rationale for learning at a student's capability frontier.** (a) Already-mastered tasks offer limited new challenge. (b) Tasks far beyond the student's current reach can admit valid teacher demonstrations without providing readily learnable supervision. (c) Frontier tasks introduce demands that are attainable but not yet reliably handled, motivating targeted skill extension. The terminal tasks are schematic examples of student-relative difficulty, not benchmark measurements.

while Evol-Instruct (Xu et al., 2024) progressively increases instruction complexity and diversity. For interactive agents, this paradigm has extended toward executable environment construction: SWE-Gym (Pan et al., 2025) provides verified software-engineering environments for training, while RST (Li et al., 2026a) recursively modifies reference solutions and aligns instructions, environments, and verifiers to generate increasingly complex terminal workflows. However, for compact students, valid and increasingly challenging tasks may still provide limited training value when their requirements exceed what the model can currently handle.

This mismatch motivates a student-relative view of task difficulty. As illustrated in Figure 1, already-mastered tasks provide limited additional challenge, while tasks far beyond the student's current reach may yield valid demonstrations that remain difficult to exploit. Between these regimes lies the student's *capability frontier*, where successful execution is attainable but not yet consistently reliable. This perspective is closely related to curriculum learning (Bengio et al., 2009) and competence-based curriculum learning (Platanios et al., 2019), which adapt training experience to learner competence. SETA (Shen et al., 2026) extends this idea to terminal-agent training by adjusting environment difficulty according to the student's success rate. However, aggregate success provides only a coarse signal: it indicates whether a task remains accessible, but provides limited guidance on *what should change next*. Similar success rates can arise from qualitatively different interaction patterns, motivating the use of richer behavioral evidence to guide task evolution.

We therefore argue that student-conditioned environment evolution requires two complementary forms of feedback.First, repeated rollouts provide an empirical estimate of whether a task remains accessible to the current student, indicating whether further evolution is appropriate. Second, the corresponding action–observation histories reveal how the student interacts with the task, exposing weaknesses in failed attempts and strategies that already succeed. These behavioral signals can then guide semantically targeted task modifications, while execution-based validation ensures that each modification remains a valid executable task. The resulting process forms a recursive evaluation-and-evolution loop in which the student determines which task demands should be explored next, while a stronger teacher subsequently provides successful demonstrations for training.

Based on this principle, we introduce **TerminalEvo**, a student-conditioned framework for evolving executable terminal environments toward the capability frontier of compact agents. TerminalEvo combines frontier-seeking environment evolution, behavior-guided semantic editing, contract-based executable task construction, and teacher-generated supervision. Rather than increasing task complexity according to a fixed schedule, it recursively adapts both how far a task evolves and what requirements are introduced next, using complementary outcome and behavioral signals from the

target student's own interactions. In this way, student outcomes regulate the extent of evolution, while action–observation trajectories guide its semantic direction.

We evaluate TerminalEvo across multiple model scales and benchmarks under approximately matched SFT supervision budgets measured in loss-bearing tokens. Compared with existing agent-training datasets, TerminalEvo achieves the best performance in seven of nine model–benchmark settings and ranks second in the remaining two, with particularly consistent gains for the 9B and 27B students. The improvements span both terminal-interaction tasks and repository-level software repair, indicating that the resulting supervision remains effective across different agent settings. Additional DPO on TerminalEvo-trained checkpoints further improves all nine scores, showing that the generated trajectories can also support subsequent preference optimization.

2 Related Work

Synthetic supervision and learner adaptation. Synthetic data has been widely used to expand supervision beyond manually curated examples. Self-Instruct bootstraps instruction-following data from a small human seed set (Wang et al., 2023), while Evol-Instruct progressively increases instruction complexity and diversity (Xu et al., 2024). Related work further studies synthetic supervision for reasoning, code generation, and agent training (Zelikman et al., 2022; Zeng et al., 2024). For compact models, the usefulness of synthetic data also depends on its suitability for the target learner. Textbooks Are All You Need emphasizes high-quality instructional data (Gunasekar et al., 2023), while Orca 2 adapts reasoning strategies to student capabilities (Mitra et al., 2023). TerminalEvo extends this learner-dependent perspective to executable agent environments.

Executable environment synthesis. Recent work has moved from static instruction–response generation toward executable environments that support tool use and multi-step interaction (Yang et al., 2026; Wu et al., 2026; Wang et al., 2026). RST (Li et al., 2026a) recursively constructs terminal tasks by extending reference solutions, aligning task components, and validating generated environments through execution. TerminalEvo builds on this executable task formulation, while focusing on how target-student behavior can guide task evolution. In this setting, execution validates whether a task is well formed, whereas student rollouts provide evidence about its difficulty and potential training value.

Difficulty adaptation and behavioral feedback. Learner-dependent difficulty has been studied in curriculum learning and adaptive training (Bengio et al., 2009; Matiisen et al., 2019). In agent training, SETA (Shen et al., 2026) adapts environment evolution according to the training model's success rate, while trajectory-level signals have also been used for diagnosis, reflection, and agent improvement (Gao et al., 2026; Shinn et al., 2023). TerminalEvo combines these two signals: rollout outcomes determine whether further evolution is appropriate, while action–observation trajectories are analyzed to guide the semantic direction of subsequent task transformations.

3 TerminalEvo

To operationalize the student-conditioned environment evolution described above, we introduce **TerminalEvo**. The framework uses outcome feedback from a fixed target student to regulate whether task evolution continues, while behavioral feedback guides the semantic direction of subsequent transformations. Validated tasks are repeatedly evaluated by the same student throughout environment generation, after which a stronger teacher produces demonstrations for student training. Figure 2 summarizes the overall framework.

3.1 Problem Formulation

Executable tasks and student interaction. An executable terminal task is represented as

$$\tau = (I, E, S, V), \tag{1}$$

where I is the public instruction, E the initial executable environment, S a reference solution, and V a verifier. The student interacts only with the public interface $\tau^{\text{pub}} = (I, E)$; the reference solution and verifier are used for task construction and evaluation but are not exposed to the student.

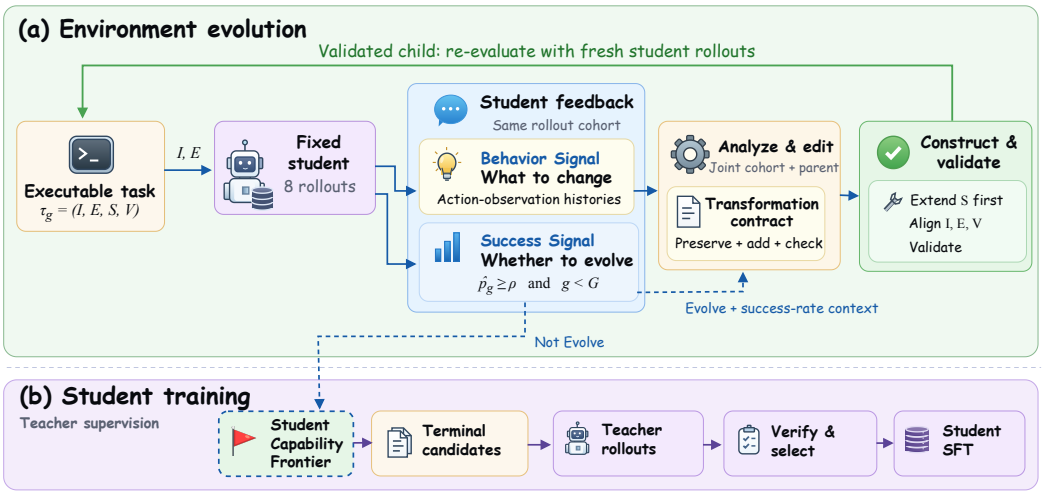

Figure 2: **Overview of TerminalEvo.** (a) Eight rollouts from a fixed student provide complementary feedback: interaction histories guide a transformation contract, while success counts determine whether the task remains eligible for evolution. The reference solution is extended first, followed by alignment of the instruction, environment, and verifier. A validated child is evaluated with fresh student rollouts. (b) After evolution, a stronger teacher generates demonstrations on terminal candidates; verification and selection precede student SFT. The capability frontier denotes the student-relative target of environment evolution, rather than a guarantee for every candidate.

Under a fixed inference and interaction budget B, a student policy π_θ produces a recorded interaction history ξ consisting of its actions and available observations. Let T be the number of actions and $x_T(\xi)$ the environment state after the final action. For the binary outcomes used during environment generation, task success is

$$r(\tau, \xi) = V(x_T(\xi)) \in \{0, 1\}, \tag{2}$$

where $r = 1$ indicates successful completion. This verification outcome is distinct from the observations available to the student during execution.

Student-relative difficulty. Task difficulty depends on the student and its interaction configuration. We characterize task accessibility by the success probability

$$p_{\theta,B}(\tau) = \Pr_{\xi \sim \pi_\theta(\tau^{\text{pub}}; B)} [r(\tau, \xi) = 1]. \tag{3}$$

A validated reference solution establishes a route to success under the verifier, but does not establish that the student can find it. We use *empirical capability frontier* to describe tasks for which repeated student rollouts provide evidence that success is attainable but not yet reliable.

Student-conditioned data generation. Given validated seed tasks and a target student, our objective is to construct environments that yield useful supervision under a limited training budget. Student rollouts guide environment design, while a stronger teacher supplies successful demonstrations. The resulting task pool $\mathcal{T}_\theta$ and training corpus $\mathcal{D}_\theta$ are constructed separately for each target student. The two design decisions are therefore how far to evolve a task and which task requirements the student should practice next.

3.2 Frontier-Seeking Environment Evolution

For each validated task τ_g at generation g, we collect K rollouts using the same student checkpoint and interaction configuration. Let

$$\Xi_g = (\xi_{g,1}, \ldots, \xi_{g,K}), \qquad R_g = (r_{g,1}, \ldots, r_{g,K}) \tag{4}$$

denote the recorded histories and their outcomes, where $r_{g,i} = r(\tau_g, \xi_{g,i})$. The empirical estimate of $p_{\theta,B}(\tau_g)$ is

$$\widehat{p}_g = \frac{1}{K} \sum_{i=1}^{K} r_{g,i}. \tag{5}$$

We use $K = 8$ and an expansion threshold $\rho = 0.5$. A task remains eligible for further evolution when

$$\widehat{p}_g \geq \rho \qquad \text{and} \qquad g < G, \tag{6}$$

where G is the maximum evolution depth. Thus, at least four successes in eight attempts are required for expansion, provided the depth limit has not been reached.

When both conditions hold, τ_g serves as the parent for the next transformation. Otherwise, the evolution branch terminates and the current validated task becomes a terminal candidate for subsequent supervision. Nonterminal intermediate tasks are not directly retained as candidates. Importantly, the threshold controls recursive expansion rather than certifying frontier membership: both zero-success tasks and tasks that remain easy when the depth limit is reached may terminate as candidates.

The same rollout cohort provides complementary signals for the next decision. The aggregate success rate determines whether evolution continues, while the corresponding interaction histories provide behavioral evidence for determining the semantic direction of the next transformation.

3.3 Behavior-Guided Task Transformation

For a parent eligible for expansion, the behavioral analyzer $\mathcal{A}$ receives the task context τ_g^{ctx}, complete recorded action–observation histories Ξ_g, individual outcomes R_g, and their aggregate success rate. It produces

$$z_g = \mathcal{A}\big(\tau_g^{\mathrm{ctx}}, \Xi_g, R_g, \widehat{p}_g\big)\,, \tag{7}$$

where τ_g^{ctx} denotes the task information supplied for analysis and z_g is the resulting behavioral analysis. The analysis is passed to the editor together with the parent task. This separates interpretation of the interaction histories from the specification of an executable change.

Learning from failures and extending successful strategies. Failed attempts provide evidence of possible weaknesses through the relationship between the student's actions, available observations, and task requirements. For example, acting before inspecting relevant state may motivate practice in evidence-dependent decisions, while an incorrect update may suggest difficulty coordinating constraints. Successful attempts reveal strategies that suffice for the parent task and can motivate new dependencies or interacting requirements that demand an extension of those strategies. The analyzer considers all rollouts jointly, without assigning a fixed failure-first or success-first priority to mixed outcomes. Its interpretation is a hypothesis about useful practice, not an established causal diagnosis of student failure.

Given the parent task and analysis, the semantic editor $\mathcal{G}$ produces a transformation contract

$$C_{g+1} = \mathcal{G}(\tau_g, z_g). \tag{8}$$

The contract specifies the parent requirements to preserve, the new requirement motivated by the behavioral evidence, and the checks for the resulting task. It therefore serves as the specification for constructing the executable child task. The analyzer and editor denote functional roles and need not correspond to different underlying models.

3.4 Executable Task Construction and Validation

We construct the child task according to C_{g+1}, following the solution-first procedure of RST (Li et al., 2026a). We first extend the reference solution, then align the public instruction, executable environment, and verifier with the inherited and newly introduced requirements:

$$\tau_{g+1} = (I_{g+1}, E_{g+1}, S_{g+1}, V_{g+1}). \tag{9}$$

The new requirement and the information needed to satisfy it must be accessible through the public task interface.

Execution-based validation checks the constructed package before further student evaluation. Reference execution checks that the revised solution passes the verifier. Negative controls test whether specified requirement violations are rejected because of those violations, rather than unrelated execution errors. Additional consistency and runtime checks examine agreement among the task components, malformed artifacts, unintended no-op behavior, and exposure of private components. Candidates with repairable defects are repaired and revalidated. A child that does not pass validation is neither evaluated by the student nor admitted to the collection pool.

Once validated, the child returns to student evaluation with fresh rollouts, closing the recursive evolution loop. The resulting outcomes again determine whether evolution continues, while the corresponding interaction histories provide behavioral evidence for the next transformation. Tasks for which evolution terminates are passed to the teacher-supervision stage described next.

3.5 Task Collection and Teacher Supervision

After environment generation, teacher collection operates on the validated terminal candidates in $\mathcal{C}_\theta$, selected by the stopping rule in Section 3.2. Student assessment histories serve as diagnostic evidence, not as SFT targets. For each candidate τ, a stronger teacher generates multiple trajectories. Let $\Xi^{\text{teacher}}(\tau)$ denote the collected trajectories and define their verifier-successful subset as

$$\Xi^{+}_{\text{teacher}}(\tau) = \left\{ \xi \in \Xi^{\text{teacher}}(\tau) \;\middle|\; r(\tau, \xi) = 1 \right\}. \tag{10}$$

Candidates with no successful teacher trajectory are discarded. This is a collection-based criterion, not a claim that the task is intrinsically unsolvable.

Final selection checks semantic validity, public availability of requirements, and substantive changes from parent tasks, together with holdout separation, content and trajectory-identity deduplication, command-signature deduplication, and foundation-quota selection. The retained task pool therefore satisfies

$$\mathcal{T}_\theta \subseteq \left\{ \tau \in \mathcal{C}_\theta \;\middle|\; \Xi^{+}_{\text{teacher}}(\tau) \neq \varnothing \right\}. \tag{11}$$

Every retained task meets the stopping and admission criteria, but this does not establish that every task is solvable yet unreliable for the student.

For each retained task, let $\Xi^{\text{SFT}}(\tau)$ be the nonempty subset of successful teacher trajectories retained for supervision. The SFT corpus is

$$\mathcal{D}_\theta = \left\{ (\tau, \xi) \;\middle|\; \tau \in \mathcal{T}_\theta,\ \xi \in \Xi^{\text{SFT}}(\tau) \right\}. \tag{12}$$

Here, τ identifies the task associated with a demonstration, not the serialized student input. Unsuccessful teacher trajectories are retained separately as a source for subsequent DPO data construction; they are not SFT targets. Environment evolution, teacher collection, and final selection are performed separately for Qwen3.5-4B, 9B, and 27B, yielding a distinct task pool and corpus for each student. The SFT and subsequent DPO experiments are described in Section 4.

4 Experiments

We evaluate TerminalEvo across three model scales and three benchmarks. We first describe the evaluation setting and training corpus, then compare SFT datasets under approximately matched supervision budgets and assess subsequent DPO gains. Section 4.5 presents a brief evolution case; Appendix C provides supplementary experimental analyses.

4.1 Evaluation Setting

Models and benchmarks. We evaluate Qwen3.5-4B, Qwen3.5-9B, and Qwen3.5-27B on Long-Horizon Terminal-Bench (LHTB) (Li et al., 2026b), Terminal-Bench 2.1 (Merrill et al., 2026a), and SWE-bench Verified (Jimenez et al., 2024). These benchmarks cover long-horizon terminal interaction, general executable terminal tasks, and repository-level software repair, respectively. We use the original model checkpoints without task-specific training as the *Vanilla* baseline.

Baselines. We compare TerminalEvo with four agent-training datasets: RST (Li et al., 2026a), TMax (Ivison et al., 2026), OpenThoughts-Agent (OT-Agent) (Raoof et al., 2026), and SETA (Shen et al., 2026). Appendix B.1 describes the source selections and preprocessing conventions.

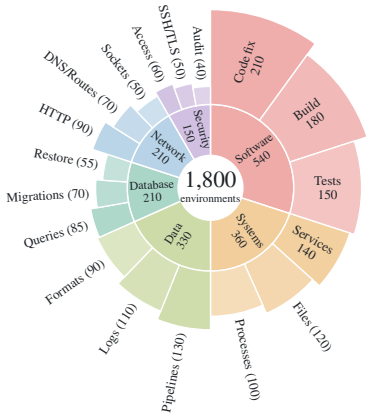

Figure 3: **Task-domain distribution.** Counts for six domains and eighteen subdomains. Angles encode domain shares; radial extensions encode subdomain counts.

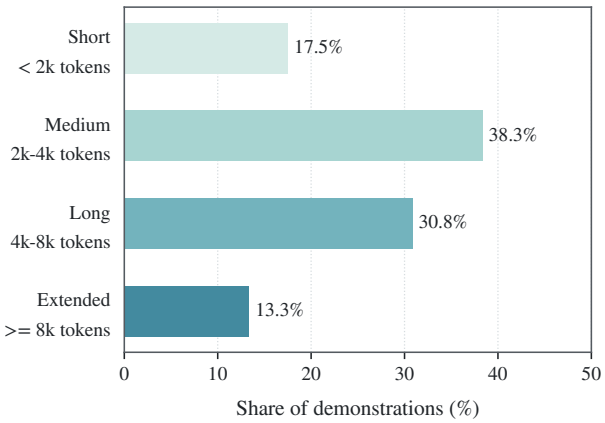

Figure 4: **Trajectory-length distribution.** Shares of complete teacher demonstrations before packing or truncation. Sequence length includes masked context; it is not the SFT loss-bearing token budget. Counting rules are in Appendix C.1.

Evaluation metrics. We report mean reward on LHTB, task success rate on Terminal-Bench 2.1, and resolved-instance percentage on SWE-bench Verified. LHTB scores and gains in the main text and tables use a 100-fold reward scale; Appendix C.4.2 plots the corresponding unit-scale rewards. All methods within each benchmark are evaluated using the same task set, agent interface, and inference budget. Appendix B details the evaluation metrics and comparison protocol.

4.2 Training Corpus Characterization

We characterize the supervision generated by TerminalEvo along three complementary dimensions: task domain, demonstration length, and terminal skill. Figure 3 shows that the generated tasks span six domains and eighteen subdomains, covering a broad range of terminal workflows. Figure 4 summarizes the distribution of teacher-demonstration lengths, showing that the corpus contains both short interactions and substantially longer multi-step trajectories. Together, these two views characterize the breadth and interaction complexity of the generated supervision. Lengths are measured on complete teacher demonstrations before packing or truncation, whereas the matched SFT budgets in Section 4.3 count only loss-bearing tokens.

Appendix C.2 complements these distributions with a breakdown of primary terminal skills by demonstration length and target student.

4.3 SFT under Matched Supervision Budgets

We compare training-data utility through supervised fine-tuning (SFT). Within each model size, all conditions share the same initialization and SFT configuration, with the dataset as the only experimental variable. Loss-bearing token counts are approximately matched to control effective supervision volume. Appendix B.3 defines token exposure and the matching criterion.

As shown in Table 1, TerminalEvo achieves the best performance on all three benchmarks for both Qwen3.5-9B and Qwen3.5-27B. For Qwen3.5-9B, it outperforms the strongest competing dataset by 0.77 on LHTB, 0.69 percentage points on Terminal-Bench 2.1, and 1.05 percentage points on SWE-bench Verified. For Qwen3.5-27B, the corresponding gains are 0.34, 0.30, and 0.51, respectively. The improvements across both terminal-interaction benchmarks and repository-level software repair indicate that the resulting supervision remains effective across different agent tasks.

For Qwen3.5-4B, TerminalEvo ranks first on Terminal-Bench 2.1 and second on both LHTB and SWE-bench Verified. Relative to Vanilla, it improves LHTB by 3.73, Terminal-Bench 2.1 by 6.26 percentage points, and SWE-bench Verified by 3.29 percentage points. Thus, TerminalEvo pro-

Table 1: SFT performance under approximately matched loss-bearing token budgets within each model size. LHTB reports mean reward $\times 100$; TB and SWE denote Terminal-Bench 2.1 and SWE-bench Verified, both in percent. Avg. is the unnormalized arithmetic mean of the three scores. Higher is better; best and second-best results are **bold** and underlined, respectively.

	Qwen3.5-4B				Qwen3.5-9B				Qwen3.5-27B			
Datasets	LHTB	TB	SWE	Avg.	LHTB	TB	SWE	Avg.	LHTB	TB	SWE	Avg.
Vanilla	5.35	17.53	54.02	25.63	10.29	27.41	61.37	33.02	17.92	40.78	72.33	43.68
OT-Agent	7.21	20.06	55.88	27.72	15.02	32.15	62.11	36.43	22.54	43.02	<u>74.91</u>	46.82
TMax	8.34	<u>23.42</u>	**57.63**	29.80	14.37	31.26	62.98	36.20	21.76	45.06	73.64	46.82
RST	8.97	22.11	56.47	29.18	<u>15.06</u>	<u>34.02</u>	63.44	<u>37.51</u>	<u>23.52</u>	<u>45.91</u>	74.02	<u>47.82</u>
SETA	**9.81**	23.36	57.14	**30.10**	14.21	33.18	<u>64.26</u>	37.22	22.47	45.73	74.63	47.61
TerminalEvo	<u>9.08</u>	**23.79**	<u>57.31</u>	<u>30.06</u>	**15.83**	**34.71**	**65.31**	**38.62**	**23.86**	**46.21**	**75.42**	**48.50**

Table 2: **DPO gains over TerminalEvo SFT.** Scores after SFT $\rightarrow$ DPO and improvements (Δ) over Table 1. LHTB uses mean reward multiplied by 100; TB2.1 and SWE-V use percentages, with gains in percentage points. No RL stage is used.

	LHTB ↑		TB2.1 ↑		SWE-V ↑	
Model	DPO	Δ	DPO	Δ	DPO	Δ
Qwen3.5-4B	10.22	+1.14	25.60	+1.81	57.44	+0.13
Qwen3.5-9B	16.07	+0.24	35.89	+1.18	65.90	+0.59
Qwen3.5-27B	25.71	+1.85	47.55	+1.34	76.02	+0.60

vides substantial gains over the original checkpoint even at the smallest model scale, although its advantage over competing training datasets is less consistent than for the 9B and 27B models.

Overall, TerminalEvo ranks first in seven of nine model–benchmark settings and second in the remaining two under approximately matched supervision budgets. These results show that TerminalEvo-generated supervision is consistently competitive across model capacities and transfers across both terminal interaction and repository-level software repair.

4.4 DPO ON TERMINALEVO SFT CHECKPOINTS

We examine whether TerminalEvo-generated trajectories provide useful preference supervision beyond SFT. Direct preference optimization (DPO) starts from each TerminalEvo SFT checkpoint and uses only preferences constructed from these trajectories, with no additional RL stage. Table 2 reports the resulting scores and gains over SFT.

DPO improves all nine model–benchmark scores. Terminal-Bench 2.1 gains range from 1.18 to 1.81 percentage points, whereas SWE-bench Verified gains are smaller, at 0.13–0.60 points. The benefit thus extends to repository-level repair, although its magnitude varies across benchmarks. On the two terminal benchmarks, gains do not increase monotonically with model size: the 4B model gains most on Terminal-Bench 2.1, while the 27B model gains most on LHTB (1.85 on the reported reward scale). Thus, improvements are not confined to the smallest student; the strongest SFT checkpoint also benefits from preference supervision.

Together, these results show that TerminalEvo supports both demonstration-based learning and subsequent preference optimization. The comparison includes additional preference supervision and does not establish a cross-dataset DPO advantage or a benefit at fixed total training cost. Preference-token accounting is detailed in Appendix B.

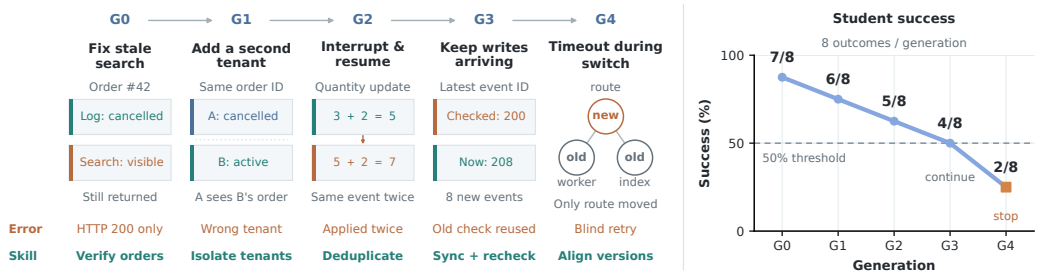

Figure 5: **An order-search evolution case.** Left: successive requirements, failure patterns, and practice targets. Right: fixed-student success over eight attempts per generation; the dashed line marks the 50% expansion threshold.

4.5 Case Study: Multi-round Environment Evolution

Figure 5 illustrates how TerminalEvo evolves a single order-search task over multiple generations using complementary behavioral and outcome feedback. The initial task exposes a stale-search failure, where the student relies on an HTTP 200 response without verifying the returned order state. Subsequent generations introduce increasingly interacting requirements: tenant isolation at G1, event deduplication under interruption and resumption at G2, synchronization with newly arriving writes at G3, and version alignment under a partial switch at G4. Each transformation is motivated by a distinct failure pattern and targets a corresponding capability, so the evolution extends previously sufficient strategies rather than adding unrelated complexity.

The right panel shows how rollout outcomes regulate this process. Using the same fixed student throughout, the number of successful executions decreases from $7/8$ at G0 to $6/8$, $5/8$, and $4/8$ as new requirements are introduced. Because the expansion rule includes equality, G3 remains eligible for further evolution at the 50% threshold. The next transformation reduces success to $2/8$ at G4, causing the branch to terminate. This example therefore illustrates the interaction between the two signals in TerminalEvo: behavioral evidence determines the semantic direction of each transformation, while empirical success controls how far the branch continues to evolve. Additional evolution cases are provided in Appendix D.

5 Discussion and Limitations

TerminalEvo introduces a student-conditioned approach to executable environment generation for compact terminal agents. A fixed target student provides two complementary signals: rollout outcomes regulate whether evolution continues, while action–observation histories guide how the task should change. The resulting transformations are instantiated through solution-first construction and execution-based validation, after which a stronger teacher generates verified demonstrations on the student-specific task pool.

Empirically, TerminalEvo provides effective supervision across model scales and evaluation settings. Under approximately matched SFT budgets, it ranks first in seven of nine model–benchmark settings and second in the remaining two, with the most consistent gains for the 9B and 27B students. The improvements span both terminal interaction and repository-level software repair, while subsequent DPO further improves all nine scores.

TerminalEvo nevertheless inherits several practical limitations common to executable agent training. Repeated student rollouts, environment construction, validation, and teacher demonstration collection introduce additional offline generation cost. The framework also relies on executable environments with sufficiently reliable verifiers, which may limit direct application to more open-ended or weakly verifiable domains. Finally, our experiments focus on terminal agents from a single model family; extending the framework to other agent architectures, tool ecosystems, and multimodal environments remains an important direction for future work.

AI USE STATEMENT

Generative AI was used to analyze student interaction histories, propose task transformations, construct executable task packages, and generate teacher demonstrations, as described in Section 3. Generated tasks undergo execution-based validation, and teacher demonstrations are selected using task verifiers (Appendix A). Generative AI tools also assisted with the interpretation and presentation of experimental results, drafting and refining manuscript text, preparing figures, and LaTeX formatting. The authors retain responsibility for the final manuscript, including its claims, interpretations, references, and AI-assisted artifacts.

REFERENCES

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Coleman Richard Charles Hooper, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. TinyAgent: Function calling at the edge. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 80–88. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-demo.9. URL https://aclanthology.org/2024.emnlp-demo.9/.

Chongyang Gao, Diji Yang, Shuyan Zhou, Xichen Yan, Luchuan Song, Shuo Li, and Kezhen Chen. Classroom final exam: An instructor-tested reasoning benchmark. *arXiv preprint arXiv:2602.19517*, 2026.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023. URL https://arxiv.org/abs/2306.11644.

Peter Hase, Mohit Bansal, Peter Clark, and Sarah Wiegreffe. The unreasonable effectiveness of easy training data for hard tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7002–7024. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.378. URL https://aclanthology.org/2024.acl-long.378/.

Hamish Ivison, Junjie Oscar Yin, Rulin Shao, Teng Xiao, Nathan Lambert, and Hannaneh Hajishirzi. Tmax: A simple recipe for terminal agents. *arXiv preprint arXiv:2606.23321*, 2026.

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

Yuetai Li, Xiang Yue, Zhangchen Xu, Fengqing Jiang, Luyao Niu, Bill Yuchen Lin, Bhaskar Ramasubramanian, and Radha Poovendran. Small models struggle to learn from strong reasoners. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 25366–25394. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.1301. URL https://aclanthology.org/2025.findings-acl.1301/.

Zhongzhi Li, Yucheng Shi, Zongxia Li, Ruhan Wang, Anhao Li, Zixun Huang, Junyao Yang, Lei Ke, Ninghao Liu, Haitao Mi, and Leowei Liang. Recursive synthesis for long-horizon terminal tasks. *arXiv preprint arXiv:2608.05466*, 2026a. URL https://arxiv.org/abs/2608.05466.

Zongxia Li, Zhongzhi Li, Yucheng Shi, Ruhan Wang, Junyao Yang, Zhichao Liu, Xiyang Wu, Anhao Li, Yue Yu, Ninghao Liu, Lichao Sun, Haotao Mi, and Leowei Liang. Long-horizon-terminal-bench: Testing the limits of agents on long-horizon terminal tasks with dense reward-based grading, 2026b. URL https://arxiv.org/abs/2607.08964.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32431–32454. PMLR, 2024b. URL https://proceedings.mlr.press/v235/liu24ce.html.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9):3732–3740, 2019.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026a. URL https://arxiv.org/abs/2601.11868.

Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Jenia Jitsev, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Kwesi Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=a7Qa4CcHak.

Arindam Mitra, Luciano Del Corro, Shweti Mahajan, Andres Codas, Clarisse Simoes, Sahaj Agarwal, Xuxi Chen, Anastasia Razdaibiedina, Erik Jones, Kriti Aggarwal, Hamid Palangi, Guoqing Zheng, Corby Rosset, Hamed Khanpour, and Ahmed Awadallah. Orca 2: Teaching small language models how to reason, 2023. URL https://arxiv.org/abs/2311.11045.

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. PMLR, 2025.

Dan Peng, Zhihui Fu, and Jun Wang. PocketLLM: Enabling on-device fine-tuning for personalized LLMs. In *Proceedings of the Fifth Workshop on Privacy in Natural Language Processing*, pp. 91–96. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.privatenlp-1.10/.

Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabas Poczos, and Tom Mitchell. Competence-based curriculum learning for neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 1162–1172. Association for Computational Linguistics, 2019. doi: 10.18653/v1/N19-1119. URL https://aclanthology.org/N19-1119/.

Negin Raoof, Richard Zhuang, Marianna Nezhurina, Etash Guha, Atula Tejaswi, Ryan Marten, Charlie F. Ruan, Tyler Griggs, Alexander Glenn Shaw, Hritik Bansal, E. Kelly Buchanan, Artem Gazizov, Reinhard Heckel, Chinmay Hegde, Sankalp Jajee, Daanish Khazi, Emmanouil Koukoumidis, Xiangyi Li, Hange Liu, Shlok Natarajan, Harsh Raj, Nicholas Roberts, Ethan Shen, Nishad Singhi, Michael Siu, Ashima Suvarna, Hanwen Xing, Patrick Yubeaton, Robert Zhang, Leon Liangyu Chen, Xiaokun Chen, Steven Dillmann, Saadia Gabriel, Xunyi Jiang, Anurag Kashyap, Boxuan Li, Yein Park, Minh Pham, Sujay Sanghavi, Lin Shi, Ke Sun, Yixin Wang, Zhiwei Xu, Erica Zhang, Siyan Zhao, Wanjia Zhao, Jenia Jitsev, Alex Dimakis, Benjamin Feuer, and Ludwig Schmidt. OpenThoughts-Agent: Data recipes for agentic models. *arXiv preprint arXiv:2606.24855*, 2026. URL https://arxiv.org/abs/2606.24855.

Qijia Shen, Zhiqi Huang, Vamsidhar Kamanuru, Aznaur Aliev, Jay Rainton, Ahmed Awelkair, Zhichen Zeng, Jiajun Li, Shi Dong, Yueming Yuan, Boyuan Ma, Qizheng Zhang, Jiwei Fu, Yuzhen Mao, Wendong Fan, Ping Nie, Philip Torr, Bernard Ghanem, Changran Hu, Jonathan Lingjie Li, Urmish Thakker, and Guohao Li. SETA: Scaling environments for terminal agents. *arXiv preprint arXiv:2607.10891*, 2026. URL https://arxiv.org/abs/2607.10891.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Segev Shlomov, Alon Oved, Sami Marreed, Ido Levy, Offer Akrabi, Avi Yaeli, Łukasz Strąk, Elizabeth Koumpan, Yinon Goldshtein, Eilam Shapira, Nir Mashkif, and Asaf Adi. From benchmarks to business impact: Deploying IBM generalist agent in enterprise production. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(47):40423–40431, 2026. doi: 10.1609/aaai.v40i47.41485. URL https://ojs.aaai.org/index.php/AAAI/article/view/41485.

Guokai Tang and Feng Zhao. SmartAD: Capacity-aligned agent distillation for small language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 27045–27057. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1349. URL https://aclanthology.org/2026.findings-acl.1349/.

Tianyou Wang, Chongyang Gao, Kezhen Chen, Chen Dong, Yinghao He, Donghan Li, Wangcheng Xu, Hongjiu Zhang, and Chi Li. Fm-bench: A benchmark for long-horizon management with competing agents. *arXiv preprint arXiv:2608.18423*, 2026.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754/.

Siwei Wu, Yizhi Li, Yuyang Song, Wei Zhang, Yang Wang, Riza Batista-Navarro, Xian Yang, Mingjie Tang, Bryan Dai, Jian Yang, et al. Large-scale terminal agentic trajectory generation from dockerized environments. *arXiv preprint arXiv:2602.01244*, 2026.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2304.12244.

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

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

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*, pp. 3053–3077, 2024.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. LIMA: Less is more for alignment. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/ac662d74829e4407ce1d126477f4a03a-Abstract-Conference.html.

A Task Validation and Supervision Selection

This appendix complements Section 3 by detailing the criteria for task validation and demonstration selection. Executability, student-relative difficulty, and the availability of successful teacher demonstrations are distinct properties: evidence for one does not establish the others.

A.1 Consistency of task requirements

Each transformation preserves the parent task's requirements and introduces an additional requirement motivated by the behavioral analysis. The public instruction, initial environment, reference solution, and verifier must specify the same inherited and added requirements. In particular, the information needed to satisfy a new dependency must be accessible through the public task interface; the reference solution and private grading artifacts are not student inputs.

The behavioral analysis considers successful and unsuccessful trajectories jointly. Failures can motivate practice on an unmet requirement, while successful strategies can motivate extensions that require additional coordination. Neither outcome type has a fixed priority. These interpretations guide task design but do not, by themselves, establish the cause of a student's failure or the learning value of the proposed extension. The revised task is therefore validated and evaluated through fresh student rollouts.

A.2 Execution-based validation

Validation examines both the feasibility of the task and the verifier's response to specified requirement violations. Executing the reference solution in a fresh environment tests whether it produces an accepted outcome without relying on state left by an earlier execution. The resulting files and outputs are checked against the declared requirements. A complementary no-op test examines whether the initial environment already passes verification without the required work.

Negative controls introduce specific violations and test whether the verifier rejects them for the intended reason, rather than because of unrelated execution errors. Repaired tasks undergo validation again, and tasks that fail validation are excluded from subsequent student evaluation and demonstration collection. Together, these checks provide evidence of executability and consistency. They do not establish that the verifier accepts every valid strategy or rejects every invalid outcome.

A.3 Selection of teacher supervision

Teacher demonstrations are collected from validated tasks at which evolution stops: the student succeeds in fewer than four of eight attempts, or the maximum evolution depth is reached. Nonterminal intermediate tasks are not directly included. A stronger teacher generates multiple trajectories for each candidate; at least one verifier-successful trajectory is required for retention. Successful teacher trajectories supply SFT targets, while unsuccessful trajectories are retained separately as a source of preference data. Student assessment trajectories guide environment evolution and are not used as SFT targets.

Final selection considers semantic validity, the public availability of task requirements, substantive changes from the parent, and separation from the holdout set. Deduplication considers task content, trajectory identity, and command signatures, followed by quota-based selection. Consequently, the training corpus is a selected subset of successful teacher demonstrations on validated terminal candidates, rather than a record of every task visited during evolution.

The stopping rule does not guarantee that each retained task has a true student success probability near 50%. Candidates may include tasks with no observed student successes and tasks that remain easy at the depth limit. Environment generation and teacher collection are performed separately for the 4B, 9B, and 27B students, with each student checkpoint fixed throughout generation. The resulting corpora are therefore student-specific; the analyzer, editor, and teacher denote functional roles and need not use different underlying model families.

Table 3: Baseline training-data sources and selection criteria. These conditions define the supervision compared with TerminalEvo in Table 1.

Baseline	Training data and selection
RST	Recursive Task Synthesis trajectories, cap10 configuration
TMax	`allenai/tmax-sft`, successful trajectories only
OpenThoughts-Agent	OpenThoughts-Agent-v1-SFT
SETA	Kimi K2.5 thinking trajectories with source reward equal to one

B Experimental Protocol

The experiments evaluate two questions: the utility of different training datasets under approximately matched SFT supervision budgets, and the incremental benefit of DPO applied to TerminalEvo SFT checkpoints. We use Qwen3.5-4B, 9B, and 27B, with a separately generated TerminalEvo corpus for each student. Within each model size, the SFT comparisons share the model initialization and optimization configuration described in Section 4.3.

B.1 Baseline data and preprocessing

Table 3 summarizes the four baseline data conditions. RST uses the cap10 configuration. TMax uses the trajectory dataset `allenai/tmax-sft`, rather than a task-only environment release. The Vanilla baseline is the original checkpoint without task-specific training.

Preprocessing preserves the source-specific interaction formats: TMax uses native bash-tool supervision, SETA retains its six-tool protocol, and the other baseline sources use Terminus-style action messages. A shared token-level training representation therefore does not imply identical action spaces in the source datasets. These differences are part of the training-data conditions; within each evaluation benchmark, all conditions use the same agent interface and inference budget, as specified in Section 4.1.

Success-based filtering uses the source dataset's reward annotations and should not be interpreted as independent re-execution of every retained trajectory. Likewise, train–holdout separation during preprocessing does not by itself establish the absence of overlap with the evaluation benchmarks.

B.2 Training configuration

Table 4 summarizes the default optimization settings of the RST-Train implementation. Both stages use a warmup phase followed by cosine learning-rate decay, with the minimum learning rate set to one tenth of the peak. SFT uses a global batch of 128 demonstrations. The DPO configuration accumulates four preference pairs per data-parallel worker, corresponding to $4W$ pairs per complete optimizer step with W workers. Training exposure is measured separately by consumed loss-bearing tokens, as defined below.

The SFT implementation computes the loss on unmasked assistant tokens, while instructions and environment observations remain conditioning context. The training configuration rejects sequences exceeding the length limit rather than silently truncating them. The DPO implementation uses AdamW with a maximum gradient norm of 1.0, preference coefficient $\beta = 0.1$, and no label smoothing. Its length-normalized objective is specified in Appendix B.5.

B.3 Accounting for supervised tokens

We measure SFT supervision by the number of consumed loss-bearing tokens. For model size m and data condition d, let $\mathcal{B}_{m,d,s}$ be the multiset of examples consumed globally at optimizer step s. For example i, let L_i denote its rendered length and $M_{i,t} \in \{0, 1\}$ indicate whether position t

Table 4: Default optimization settings in the RST-Train implementation. Sequence length includes the complete interaction context; for DPO, the limit applies separately to each trajectory in a preference pair.

Hyperparameter	SFT	DPO
Peak learning rate	3×10^{-6}	5×10^{-7}
Minimum learning rate	3×10^{-7}	5×10^{-8}
Learning-rate decay	Cosine	Cosine
Warmup fraction	0.03	0.10
Adam momentum coefficients	$(0.9, 0.98)$	$(0.9, 0.95)$
Weight decay	0.1	0
Maximum sequence length	32,768	32,768

contributes to the training loss. Cumulative supervision is

$$U_{m,d} = \sum_{s=1}^{S_{m,d}} \sum_{i \in \mathcal{B}_{m,d,s}} \sum_{t=1}^{L_i} M_{i,t}. \tag{13}$$

Prompts, observations, padding, and other masked positions contribute zero. Each consumed occurrence of a repeated or resampled example contributes to $U_{m,d}$; examples excluded before training and batches not consumed do not. The global definition counts each occurrence once across workers.

Within each model size, the data conditions use a common target budget U_m^*. The relative deviation from this target is

$$\delta_{m,d} = 100 \left(\frac{U_{m,d}}{U_m^*} - 1 \right) \%. \tag{14}$$

Matching is approximate because training consumes finite batches and a data condition may exhaust its available examples. Loss-bearing token exposure is distinct from the number of unique demonstrations, the total number of processed sequence tokens, and the cost of generating the data. Thus, matching $U_{m,d}$ controls the amount of supervised prediction without equalizing total computation.

B.4 Evaluation metrics and comparability

We evaluate long-horizon terminal interaction on LHTB, general terminal tasks on Terminal-Bench 2.1, and repository-level software repair on SWE-bench Verified. All conditions within a benchmark use the same task set, agent interface, and inference budget. LHTB is reported as mean reward; Terminal-Bench 2.1 as task success rate; and SWE-bench Verified as resolved-instance percentage. LHTB mean reward is not a percentage success rate, so its numerical changes are reported in reward units rather than percentage points.

Comparisons are made within these benchmark versions and protocols. Scores from other subsets, interaction budgets, or evaluation interfaces are not directly interchangeable. Variation across training seeds and variation across decoding trials are also distinct sources of uncertainty; the point estimates alone do not establish statistical significance.

B.5 Preference optimization after SFT

For each model size, DPO starts from the corresponding TerminalEvo SFT checkpoint and uses preference data constructed from TerminalEvo-generated trajectories. Table 2 reports the resulting checkpoints on the same three benchmarks, with Table 1 providing the SFT reference. The training sequence is SFT followed by DPO, without an additional reinforcement learning stage.

Length-normalized preference objective. The RST-Train implementation uses a sigmoid preference objective with length-normalized trajectory scores. Let $\xi = (x_1, \ldots, x_L)$ be a serialized

interaction and M_t its binary supervision mask. For a policy π_ϕ, define

$$\ell_\phi(\xi) = \frac{1}{N_\xi} \sum_{t=1}^{L} M_t \log \pi_\phi(x_t \mid x_{<t}), \qquad N_\xi = \sum_{t=1}^{L} M_t > 0. \tag{15}$$

Thus, instructions and observations condition the predictions but do not contribute directly to the score. Each trajectory is normalized by its own number of supervised tokens. Let π_{ref} be the fixed SFT reference policy and define

$$a_\phi(\xi) = \ell_\phi(\xi) - \ell_{\text{ref}}(\xi). \tag{16}$$

For a preferred and rejected pair (ξ^+, ξ^-) associated with the same task, the objective is

$$\mathcal{L}_{\text{DPO}}(\phi) = -\mathbb{E}_{(\xi^+,\xi^-)\sim\mathcal{P}_\theta} \log \sigma\big(\beta \left[a_\phi(\xi^+) - a_\phi(\xi^-)\right]\big), \tag{17}$$

where $\mathcal{P}_\theta$ denotes the preference corpus for the target student and σ is the logistic function. The reference policy remains fixed throughout optimization. Length normalization compares mean supervised-token log probabilities rather than unnormalized trajectory sums, making this choice explicit in the definition of the preference objective.

Preference supervision is additional to the approximately matched SFT budget. This experiment therefore measures the improvement from adding DPO and its associated training exposure to TerminalEvo SFT; it is neither a cross-dataset DPO comparison nor a comparison at fixed total training cost.

B.6 Complementary analyses of supervision utility

Appendix C complements the main benchmark comparisons with difficulty-stratified training results and intermediate-checkpoint learning curves. The former relate the fixed student's pre-training task accessibility to subsequent training gains; the latter characterize how supervision from each data source translates into performance over the course of SFT. These quantitative analyses assess the utility of the resulting supervision at the level of the complete generation framework.

Section 4.5 and Appendix D provide complementary process-level evidence: rollout outcomes regulate whether a task continues to evolve, while action–observation histories guide the semantic direction of each transformation. Together, the analyses and case studies connect the learning utility of TerminalEvo-generated supervision with the student-conditioned process used to construct it.

C SUPPLEMENTARY ANALYSES

We provide the counting definitions for the corpus statistics in Section 4.2 and a detailed analysis of terminal-skill composition, followed by difficulty-stratified training results and SFT learning curves that complement the benchmark comparisons in Section 4.

C.1 CORPUS STATISTICS

The corpus statistics summarize 7,200 successful teacher demonstrations. Each demonstration is counted once before packing or truncation and receives one primary skill label. The skill and length distributions describe the same set of demonstrations; records associated with different target students need not represent globally unique tasks.

Sequence length includes both supervised targets and masked context, unlike the loss-bearing token exposure in Appendix B.3. Using $1\text{k} = 1024$, the length intervals are $[1, 2048)$, $[2048, 4096)$, $[4096, 8192)$, and $[8192, \infty)$, containing 1,260, 2,760, 2,220, and 960 demonstrations, respectively. The domain distribution covers six domains and eighteen subdomains; the outer radial extensions in Figure 3 encode counts by length rather than area.

C.2 TERMINAL-SKILL COMPOSITION

Figure 6 characterizes the primary terminal skills represented in the same 7,200 successful teacher demonstrations. The eight categories cover state and log inspection, file and code editing, environment configuration, shell-pipeline composition, command-line tool use, dependency resolution, error recovery, and output verification. These labels describe the principal activity of a demonstration rather than an exhaustive inventory of its constituent actions: a trajectory assigned to error recovery may also involve inspection, editing, and verification. Assigning one primary label per demonstration makes the category counts additive without counting such actions as separate examples.



Figure 6: **Terminal-skill composition.** Upper stacks partition 7,200 successful teacher demonstrations by sequence length; lower stacks partition the same cohort by target student. Each demonstration has one primary skill. Mirrored counts are not failures or measures of student proficiency.

Length-conditioned composition. The upper stacks partition each skill category into the four sequence-length intervals defined in Appendix C.1. Their total heights measure the number of demonstrations assigned to the corresponding skill, while the colored segments distinguish shorter from more extended interactions. This view complements the aggregate length distribution by showing how interaction length varies within each skill category. Sequence length measures the extent of the recorded interaction, including masked context; it is neither a direct measure of task difficulty nor the loss-bearing token budget used in the SFT comparison.

Student-conditioned composition. The lower stacks partition those same demonstrations by the target student for which they were generated: Qwen3.5-4B, 9B, or 27B. The lower half mirrors counts for visual comparison and does not represent negative values, failed attempts, or student error rates. For each skill, the upper and lower stack magnitudes therefore agree, because they are two partitions of the same cohort. The lower segments characterize the composition of student-conditioned supervision, not the student's proficiency in the corresponding skill. Differences in these counts describe variation among the generated corpora; they do not independently identify the effect of model capacity or behavioral feedback.

The two halves report complementary marginal distributions rather than a joint cross-tabulation of skill, sequence length, and target student. They therefore support comparisons of skill prevalence across students and length composition within skills, but do not determine the length distribution for an individual student within a given skill. Together with the domain and length summaries in Section 4.2, this analysis describes the coverage and composition of the supervision used in the training experiments.

C.3 Difficulty-stratified training results

Figure 7 groups tasks by the fixed student's success count in eight pre-training attempts. Each point reports the Terminal-Bench 2.1 gain after training on the corresponding stratum, relative to the same student's original checkpoint. The grouping measures empirical accessibility for a given student rather than intrinsic task complexity. This analysis complements the framework-level comparisons by examining how student-relative task accessibility relates to the utility of the resulting supervision (Appendix B.6).

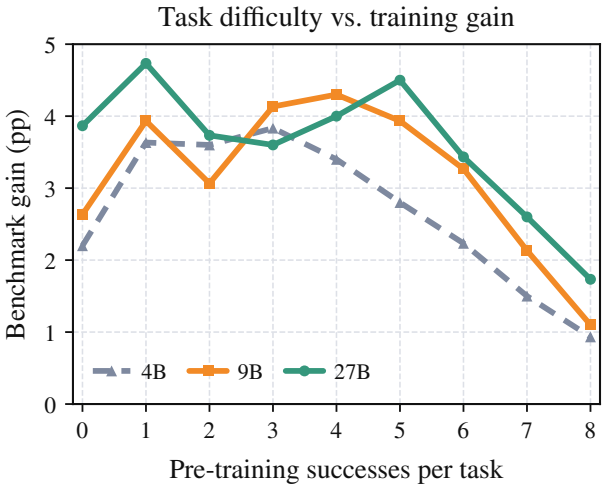

Figure 7: **Difficulty-stratified training results.** The x-axis groups tasks by a fixed student's successes in eight pre-training attempts. The y-axis reports TB2.1 gains in percentage points after training on each stratum, relative to the same model's original checkpoint.

C.4 SFT LEARNING CURVES

Figures 8–10 report SFT checkpoint evaluations on Terminal-Bench 2.1, LHTB, and SWE-bench Verified, respectively. Each curve begins at the original student checkpoint and ends at the corresponding result in Table 1, with intermediate evaluations at 10–90% of the run's final supervision budget. The horizontal axis normalizes consumed loss-bearing tokens to each run's own final budget. It therefore measures relative training progress; it does not place all runs on a shared absolute token scale or, by itself, establish absolute sample efficiency.

C.4.1 TERMINAL-BENCH 2.1

Figure 8 shows improvements over Vanilla at all three model scales. TerminalEvo reaches 23.79%, 34.71%, and 46.21% for the 4B, 9B, and 27B students, respectively, corresponding to gains of 6.26, 7.30, and 5.43 percentage points. It finishes ahead of the strongest alternative by 0.37, 0.69, and 0.30 percentage points. The final advantage is therefore consistent across scales, although individual checkpoint rankings can vary during training.

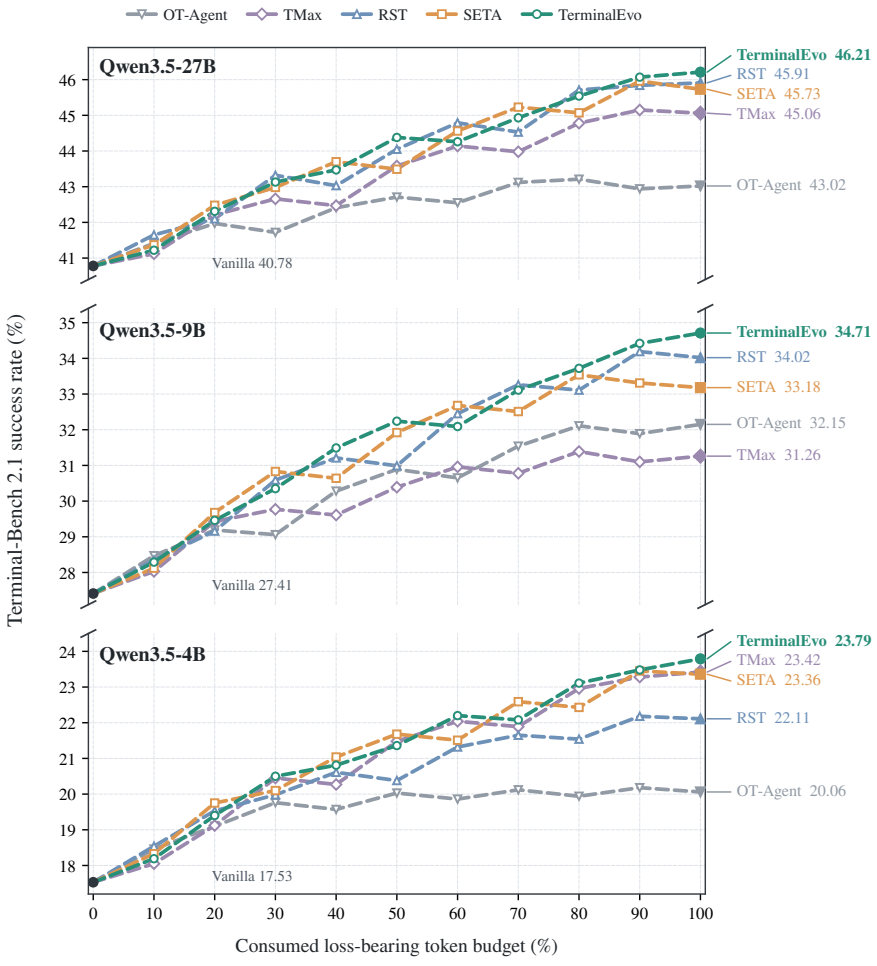

Figure 8: **SFT learning curves on Terminal-Bench 2.1.** Panels show the 27B, 9B, and 4B students from top to bottom. Vanilla and final scores match Table 1; intermediate points report checkpoint evaluations. The horizontal axis gives consumed loss-bearing tokens as a percentage of each run's own final budget.

C.4.2 LONG-HORIZON TERMINAL-BENCH

Figure 9 extends the comparison to long-horizon terminal interaction. TerminalEvo's final mean rewards are 0.0908, 0.1583, and 0.2386 for the 4B, 9B, and 27B students, improving on Vanilla by 0.0373, 0.0554, and 0.0594. These unit-scale rewards correspond to the LHTB values reported on a 100-fold scale in Table 1.

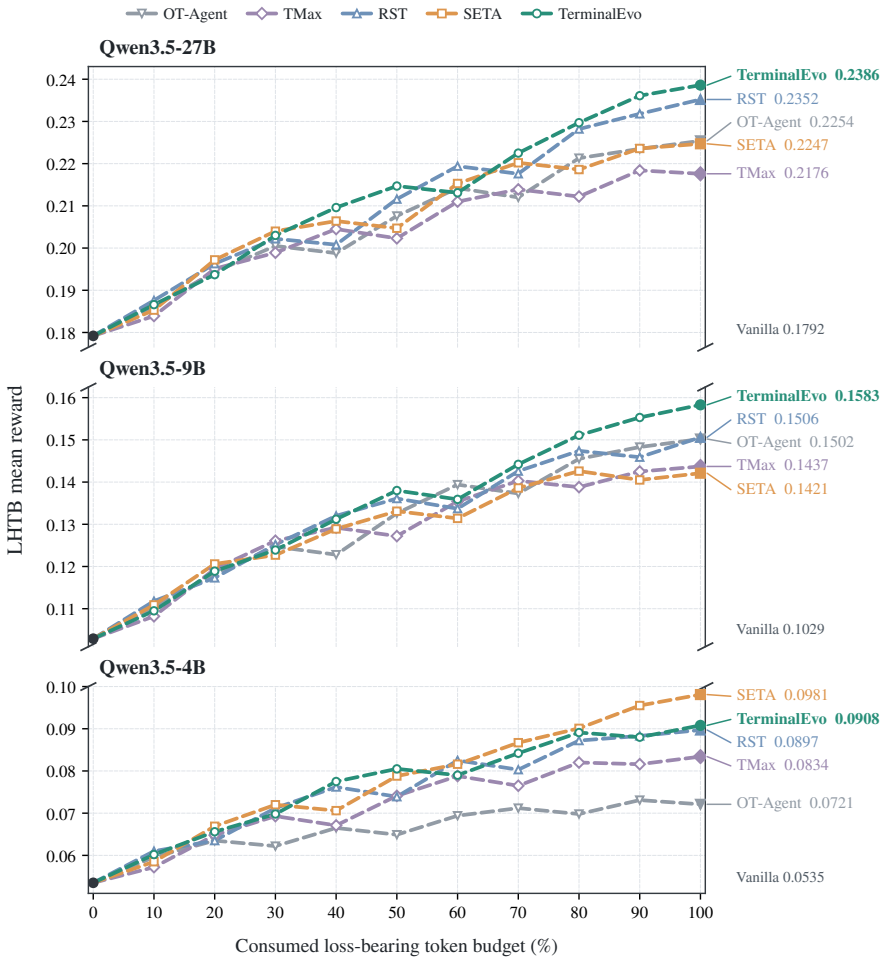

Figure 9: **SFT learning curves on Long-Horizon Terminal-Bench.** Panels show the 27B, 9B, and 4B students from top to bottom. The vertical axis reports mean reward on the unit scale; multiplying by 100 gives the LHTB scores in Table 1. The horizontal axis gives consumed loss-bearing tokens as a percentage of each run's own final budget. TerminalEvo has the highest final reward for the 9B and 27B students, while SETA leads at 4B.

The trajectories distinguish endpoint performance from uniform dominance throughout training. At 9B and 27B, TerminalEvo is not highest at every intermediate checkpoint, but its late-stage gains yield the best final rewards, exceeding RST by 0.0077 and 0.0034. At 4B, SETA continues to improve in the final portion of training and reaches 0.0981, whereas TerminalEvo finishes at 0.0908, close to RST's 0.0897. Thus, the advantage on long-horizon interaction is clearest for the two larger students; the smallest student improves substantially over Vanilla without outperforming every alternative.

C.4.3 SWE-BENCH VERIFIED

Figure 10 examines repository-level software repair. TerminalEvo reaches 57.31%, 65.31%, and 75.42% for the 4B, 9B, and 27B students, respectively. Relative to Vanilla, these are gains of 3.29, 3.94, and 3.09 percentage points, showing that the benefits of the terminal-agent supervision extend to this distinct evaluation setting.

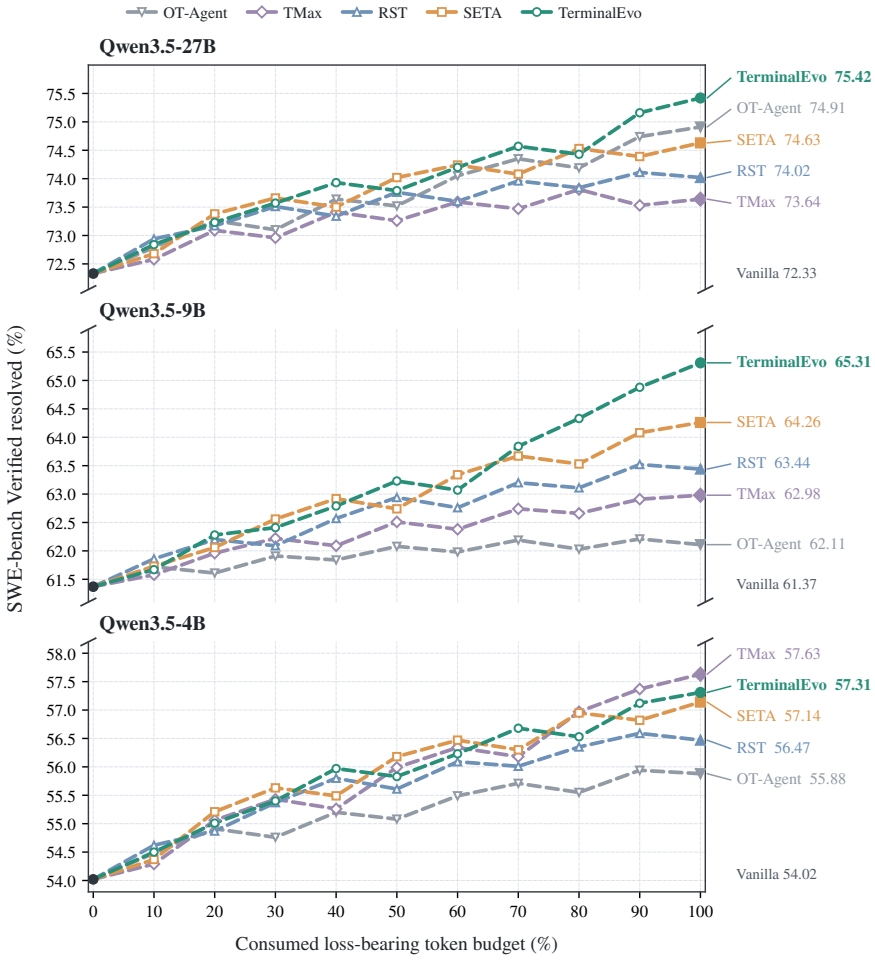

Figure 10: **SFT learning curves on SWE-bench Verified.** Panels show resolved-instance percentages for the 27B, 9B, and 4B students from top to bottom. Vanilla and final scores match Table 1. The horizontal axis gives consumed loss-bearing tokens as a percentage of each run's own final budget. TerminalEvo finishes highest at 9B and 27B; TMax has the highest final score at 4B.

For the 9B student, TerminalEvo separates from the competing curves late in training and finishes 1.05 percentage points above SETA, the strongest alternative. The 27B comparison is closer: intermediate rankings change, and TerminalEvo's final advantage over OT-Agent is 0.51 percentage points. At 4B, TMax achieves 57.63%, exceeding TerminalEvo by 0.32 percentage points. Taken together with LHTB, these curves show a more consistent final advantage at 9B and 27B than at 4B. They describe the observed optimization trajectories under the stated normalization, rather than establishing statistical significance or a universal ranking at every training stage.

D CASE STUDIES: MULTI-ROUND ENVIRONMENT EVOLUTION

Three evolution branches link successive requirements to failure patterns and practice targets. Each generation is evaluated with eight rollouts from the same fixed student. Success counts thus compare task versions, not student learning during evolution; Section 4 separately evaluates the utility of the resulting supervision.

D.1 OFFLINE BUILDS ACROSS DEPENDENCIES, ARCHITECTURES, AND CACHES

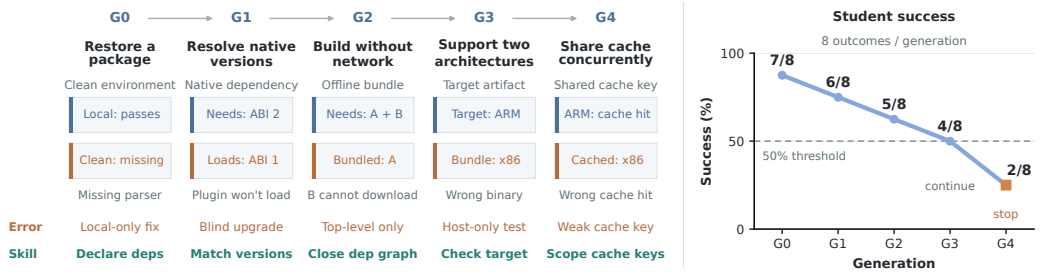

Figure 11: **Evolution of an offline-build task.** Left: dependency declaration, native-version compatibility, offline dependency coverage, target architectures, and shared-cache isolation progressively extend the task. Error patterns identify the corresponding practice targets. Right: success over eight fixed-student attempts per generation, with the 50% expansion threshold shown by the dashed line.

G0: Declare the required dependencies. The package passes locally but fails in a clean environment because a parser is missing. Local success depends on an ambient dependency that the package does not supply. The initial task therefore requires declaring the dependency so that the package can be restored independently of the original environment.

G1: Match native dependency versions. The next generation introduces a plugin requiring application binary interface (ABI) version 2, while the environment loads ABI version 1. Declaring a dependency is insufficient when the loaded native component is incompatible. The practice target shifts from dependency presence to version compatibility: a blind upgrade does not establish that the plugin's ABI requirement is met.

G2: Close the offline dependency graph. Removing network access exposes incomplete dependency coverage. The task requires A and B, but the offline bundle contains only A, leaving B unavailable when it cannot be downloaded. Correct version selection must now be combined with dependency closure: the bundle must cover the full dependency graph, rather than only top-level packages.

G3: Verify artifacts for the target architecture. Support for two architectures introduces a mismatch between an ARM target and an x86 binary in the bundle. Host-only testing can miss this incompatibility, even when all required dependencies are present. The agent must verify that the bundled native artifacts match the target, extending offline completeness with architecture-aware validation.

G4: Isolate incompatible cache entries. Concurrent builds share a cache, and an ARM request retrieves a cached x86 artifact. Selecting the correct target is no longer sufficient if cache reuse can substitute an incompatible binary. A cache hit establishes availability, not suitability. The new requirement is to scope cache keys by the relevant build context, including architecture, so that reuse preserves target compatibility.

Success counts fall from $7/8$ through $6/8$, $5/8$, and $4/8$ to $2/8$. G3 permits expansion below the depth limit; G4 stops the branch and enters the candidate pool under Equation 6.

D.2 SALES REPORTING WITH DUPLICATE, LATE, AND REVISED EVENTS

Figure 12 follows a sales-reporting pipeline whose initial numerical error evolves into interacting requirements on record identity, aggregation, event time, and revision state.

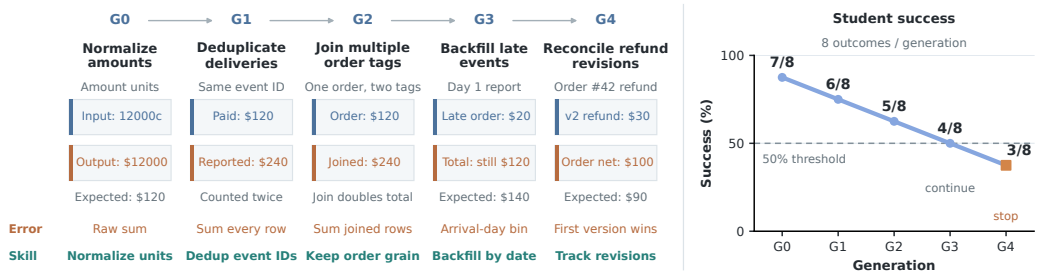

Figure 12: **Evolution of a sales-reporting pipeline.** G0–G4 successively require monetary-unit normalization, event deduplication, order-level aggregation, late-event backfilling, and refund-revision handling. The error and skill rows link each failure pattern to a practice target. The same fixed student is evaluated eight times per generation; success counts of $7/8$, $6/8$, $5/8$, $4/8$, and $3/8$ lead to continued expansion at G3 and termination at G4 under the 50% threshold.

G0: Normalize monetary units. The input amount is 12,000 cents, but the pipeline reports \$12,000 rather than \$120. Summing the raw field preserves neither the intended unit nor the numerical meaning of the report. The initial practice target is to interpret and normalize amounts before aggregation.

G1: Distinguish payments from deliveries. The next generation delivers the same event identifier more than once. A \$120 payment is then reported as \$240 when every received row is counted. Correct unit conversion alone cannot prevent the error: the pipeline must count the underlying event once, irrespective of repeated delivery. This introduces event-identity deduplication.

G2: Preserve the order-level aggregation unit. Joining an order to two tags produces two rows for a single \$120 order. Summing the joined rows again yields \$240, even if the input event stream has already been deduplicated. This generation separates duplication during ingestion from multiplicity introduced by a join. The added requirement is to preserve one monetary contribution per order while incorporating its multiple tags.

G3: Attribute late events to the correct reporting date. A \$20 order arrives after the day-one report, whose total remains \$120 instead of increasing to \$140. Assigning the event only to its arrival day leaves the original reporting period incomplete. The pipeline must use the event's relevant date to update the earlier total, while retaining the unit and identity constraints introduced in previous generations.

G4: Reconcile revised refunds. A second refund version specifies \$30 for order #42, but the reported net amount remains \$100 rather than \$90. Treating the first observed version as final fails when a later version changes the effective refund. The task therefore requires distinguishing a duplicate delivery from a semantic revision and reconciling the report with the applicable version. This is stronger than deduplication alone: repeated versions must not be double-counted, but legitimate revisions must still change the result.

The student succeeds in $7/8$, $6/8$, $5/8$, $4/8$, and $3/8$ attempts at G0–G4. The branch expands at G3 and stops at G4 under the same 50% rule. The progression shows why superficially similar total errors can require different semantic changes: unit normalization, event identity, join multiplicity, temporal attribution, and revision handling are distinct sources of inconsistency.

D.3 CERTIFICATE ROTATION ACROSS HOSTS, TRUST STATES, AND REPLICAS

Figure 13 evolves a certificate-rotation task by widening the scope of verification from a local artifact to the service configurations and clients that determine whether a connection succeeds.

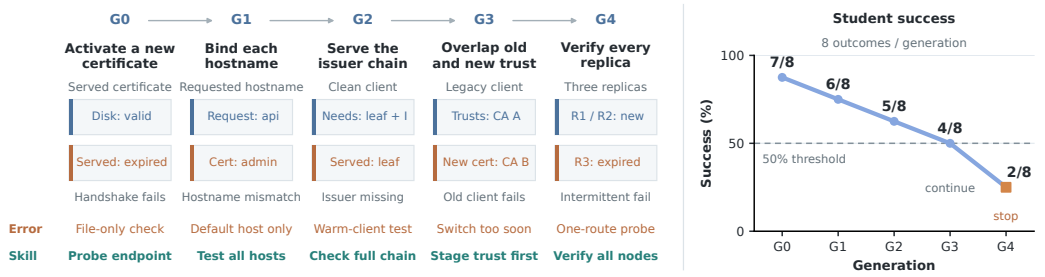

Figure 13: **Evolution of a certificate-rotation task.** G0–G4 extend verification from the served certificate to hostname bindings, issuer-chain completeness, overlapping trust states, and replica consistency. The error and skill rows distinguish the resulting failure patterns and practice targets. Success counts from eight attempts by the fixed student are $7/8$, $6/8$, $5/8$, $4/8$, and $2/8$; G3 permits expansion and G4 terminates the branch under the 50% threshold.

G0: Verify the served certificate. The certificate on disk is valid, but the endpoint still serves an expired certificate and the handshake fails. Inspecting the replacement file alone therefore does not establish successful activation. The initial task requires checking the certificate exposed by the running service, connecting the configuration change to an observable endpoint result.

G1: Check hostname-specific behavior. A request for the `api` hostname receives the certificate associated with `admin`. A successful check of the default host can miss this mismatch. The next generation adds hostname coverage to endpoint verification: each requested hostname must receive a certificate appropriate to that host.

G2: Validate the complete issuer chain. A clean client requires both the leaf certificate and an intermediate issuer, but the service supplies only the leaf. A test using a client with cached issuer information can conceal the omission. The new requirement is to validate the chain presented by the service without relying on that cached state, rather than checking the leaf certificate in isolation.

G3: Coordinate certificate and trust transitions. A legacy client trusts certificate authority A, whereas the new certificate is issued by authority B. Changing the served certificate before the client can trust B breaks the connection despite a valid hostname and issuer chain. This generation adds a transition constraint: the old and new trust states must overlap appropriately so that trust changes precede the incompatible certificate switch.

G4: Verify every replica. Replicas R1 and R2 serve the new certificate, while R3 still serves an expired one. A probe that reaches only an updated replica can report success even though connections fail intermittently. The final generation therefore extends verification from one reachable route to all replicas, requiring the agent to detect residual inconsistency across the deployment.

Success counts decrease from $7/8$ to $6/8$, $5/8$, $4/8$, and $2/8$. As in the offline-build case, G3 remains eligible for expansion and G4 terminates the branch. Across the three cases, behavioral evidence specifies which previously sufficient strategy to extend, whereas outcome feedback controls continuation. Their common downward success trend describes these branches, not a monotonicity guarantee: each validated child still requires fresh evaluation under the procedure in Section 3.2.