TRACEGYM: FAILURE-DRIVEN ENVIRONMENT SYNTHESIS FOR TRAINING TERMINAL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents are increasingly used to solve tasks in terminal environments, yet they still make frequent mistakes. These failures are a valuable learning signal, because the same kinds of errors, such as declaring success without checking the output, recur across otherwise unrelated tasks. Prior work uses such failures as feedback at inference time, fine-tunes on corrected trajectories, or synthesizes new tasks around the capability that failed. However, none of them makes the agent revisit the situation in which it erred and learn to act correctly there. We introduce TRACEGYM, a fully automated pipeline that turns an agent's failures into executable training environments, each placing the agent back in a situation where it made a wrong decision so that it can learn the right one. It has three stages. *Failure diagnosis* distills the agent's errors into *failure contracts*, reusable specifications of a wrong decision and its correct alternative. *Environment synthesis* recreates each contract in tasks drawn from a separate training pool, through a minimal change that leaves the task's objective and verifier intact. *Training* then uses this existing verifier as the only reward, so no new reward design is needed. From failures of Qwen3.6-27B on Terminal-Bench 2.1 and a task pool disjoint from the benchmark, TRACEGYM produces 225 training environments. Training the same model on these environments, without a stronger teacher, raises its Terminal-Bench 2.1 success from 48.8% to 56.2%, compared with a 4.4-point gain reported for reinforcement learning on the same pool. The gain generalizes to TerminalWorld-Verified, an out-of-distribution benchmark, where success rises from 32.0% to 37.5%, and to Qwen3.5-4B, a model whose failures were not used.

1 INTRODUCTION

LLM agents are increasingly used to solve software tasks, from resolving repository issues (Jimenez et al., 2024; Yang et al., 2024; Wang et al., 2025) to completing long command-line workflows (Merrill et al., 2026; Chu et al., 2026), and in both settings an executable verifier checks whether the task is solved. Yet agents still fail on a large share of these tasks (Jimenez et al., 2024; Merrill et al., 2026; Chu et al., 2026). These failures are informative, because a failed trajectory often shows which decision went wrong, such as declaring success without checking, trusting an unverified artifact, or stopping exploration too early. Moreover, the same kinds of errors appear repeatedly in tasks that have little else in common (Deshpande et al., 2025; Zhu et al., 2025; Cemri et al., 2026). An agent that learns to avoid an error in one task may therefore avoid it in others as well. This makes an agent's own failures a natural source of targeted practice, but turning them into training faces two obstacles. First, a failed trajectory shows the mistake but not the correct action, so there is nothing to imitate. Second, it offers only a single instance of the decision, and when the failed task comes from an evaluation benchmark, even that instance cannot be used for training.

Existing routes, compared in the bottom row of Figure 1, address only part of this problem. Reflection and diagnosis methods turn failures into feedback on prompts, workflows, or later attempts at inference time, leaving the policy unchanged (Shinn et al., 2023; Zhu et al., 2025; Cemri et al., 2026). Fine-tuning on error-correction trajectories updates the policy but never requires the agent to make the decision itself (Fu et al., 2025; Zhu et al., 2026). Environment synthesis lets the agent learn from real interaction (Su et al., 2025; Du et al., 2026) but mostly decides what to build from coverage, diversity, or difficulty rather than from the agent's own failures (Zhu et al., 2026; Pan et al., 2026; Li et al., 2026; Shi et al., 2026; Chen et al., 2026). Its failure-driven variants (Qi et al.,

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.

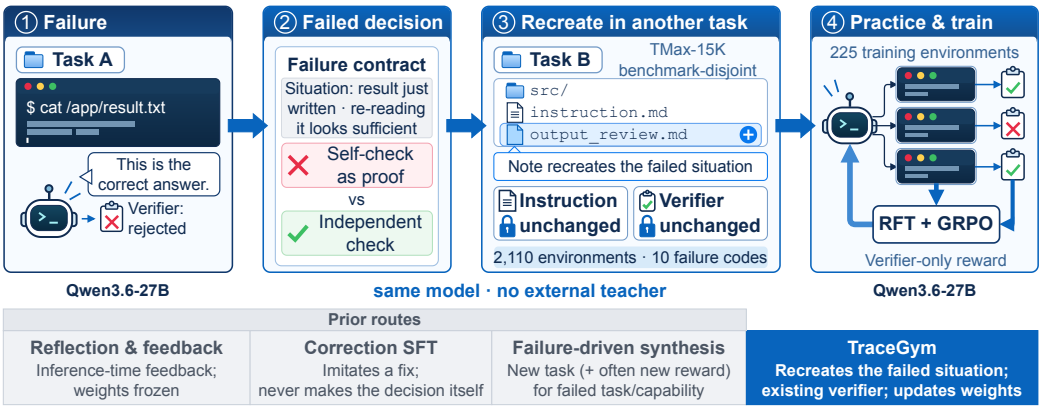

Figure 1: **Overview of TRACEGYM.** *Top:* a failed decision (①–②) is recreated in another task without editing its instruction or verifier (③) and practiced by the same model (④). *Bottom:* comparison with prior routes for learning from failures.

2025; Hao et al., 2026; Kang et al., 2026) build a new task, often with a new reward, around the task or capability that failed. None of these routes places the agent back in the situation in which it erred and rewards it for acting correctly there.

We introduce TRACEGYM, a fully automated pipeline that turns an agent's failures into executable training environments, each placing the agent back in a situation where it made a wrong decision so that it can learn the right one. The top row of Figure 1 follows one failure through the pipeline's three stages. *Failure diagnosis* groups errors into recurring failure codes and fixes, for each code, a *failure contract* that states the wrong decision and its correct alternative. In the example, the agent takes a re-read of its own output as proof of correctness (①). This invalid-verification error recurs across 30 benchmark tasks, and its contract contrasts the self-check with an independent check (②). *Environment synthesis* recreates the contract in tasks from a pool disjoint from the benchmark by adding one informational artifact, here a note that invites the same self-check, while the instruction and verifier stay unchanged, so each contract yields many practice instances outside the benchmark (③). *Training* then uses the task's own verifier as the only reward, so the model that produced the failures learns the right action from outcomes rather than demonstrations, through rejection fine-tuning (Zelikman et al., 2022; Yuan et al., 2023) followed by GRPO (Shao et al., 2024) and without an external teacher (④).

We instantiate TRACEGYM on failures of Qwen3.6-27B on Terminal-Bench 2.1, with ten failure codes and TMax-15K, a pool of 14,601 tasks disjoint from the benchmark, as the source of training tasks. Training the same model on the 225 environments where it fails with the intended code yields TRACEGYM-27B, whose Terminal-Bench 2.1 success rises from 48.8% to 56.2% (+7.4 points), compared with a gain of 4.4 points reported by Ivison et al. (2026) for reinforcement learning of the same base model on the same pool. Because the seeding failures come from Terminal-Bench 2.1, we also evaluate on TerminalWorld-Verified (Chu et al., 2026), a different task distribution, where success rises from 32.0% to 37.5%. Qwen3.5-4B, a model whose failures were not used, also improves after training on the same environments, from 17.2% to 20.2% on Terminal-Bench 2.1 and from 22.5% to 27.0% on TerminalWorld-Verified.

In summary, we make the following contributions:

- **Failure-driven environment synthesis.** We propose TRACEGYM, which turns each recurring failure code into a failure contract and recreates it in compatible tasks through a bounded change that keeps each task's verifier as the reward.
- **A fully automated pipeline.** From failures of Qwen3.6-27B on Terminal-Bench 2.1, the pipeline builds environments for ten failure codes, and training on them with rejection fine-tuning and curriculum GRPO needs neither reward design nor an external teacher.
- **Evidence of targeted, transferable gains.** Training raises Terminal-Bench 2.1 success by +7.4 points, mainly through fewer wrong answers, and the gain transfers to TerminalWorld-Verified and to Qwen3.5-4B, a model whose failures were not used.

2 Related Work

Failure trajectory analysis. Failure trajectory analysis connects recurring error patterns with the decisions that cause individual runs to fail. Taxonomy-based benchmarks make errors comparable across tasks by organizing annotated traces into reusable categories (Deshpande et al., 2025). Within a trajectory, however, attribution must identify the responsible agent or critical step (Zhang et al., 2025), while separating an initiating error from its downstream consequences. Constraint-violation evidence (Barke et al., 2026) and dependency-guided tracing (Rafi et al., 2026) provide complementary ways to ground this diagnosis in execution history. These analyses also support agent improvement: localized corrective feedback enables recovery (Zhu et al., 2025), while adaptive failure taxonomies guide candidate selection, prompt and workflow optimization, and runtime feedback (Cemri et al., 2026). TraceGym uses failure diagnosis to construct executable training environments for updating model parameters, targeting improvements in the learned policy beyond inference-time feedback and workflow optimization.

Environment synthesis for agents. Environment generation provides executable settings for such practice. For terminal agents, TermiGen constructs verifiable tasks and environments through iterative refinement, then injects errors during trajectory collection to produce recovery examples (Zhu et al., 2026). Beyond terminal tasks, CodeGym derives tool-interactive environments from coding problems (Du et al., 2026). Capability-targeted methods make desired behaviors an explicit construction input: NexForge converts high-level requirements into executable tasks and expert trajectories (Zhao et al., 2026), while TRACE contrasts successful and failed trajectories to identify capability deficits, then synthesizes environments with rewards for exercising each capability (Kang et al., 2026). Like TRACE, TraceGym uses failure evidence to target practice. It adds bounded, task-local information to a compatible existing task while preserving the source instruction, objective, and verifier. The resulting interactions retain the original task-completion reward rather than a newly defined capability-specific reward.

3 TraceGym: Failure-Driven Environment Synthesis

TraceGym is a fully automated pipeline that turns failed agent trajectories into targeted, executable practice environments in three stages: failure diagnosis, environment synthesis, and training (Figure 2). All LLM stages (diagnosis, source selection, artifact generation and re-diagnosis) use GPT-5.6-Sol (OpenAI, 2026b); Appendix D reports their token usage.

3.1 Failure diagnosis and contract abstraction

A base task $x = (u, e, v)$ consists of an instruction u, an executable environment state e, and a verifier v that maps a trajectory τ to $\{0, 1\}$. Diagnosis runs on every failed trajectory ($v(e, \tau) = 0$). Two LLM stages localize its failure points and label each with primary or secondary codes from a failure taxonomy induced from these trajectories (Section 4.1). For each selected code (below), TraceGym then fixes a single reusable *failure contract*

$$c = \big(d_{\text{err}},\ d_{\text{alt}},\ \phi_{\text{pre}},\ \mathcal{I}_{\text{inv}}\big), \tag{1}$$

pairing the erroneous decision d_{err} with the correct alternative d_{alt}, the preconditions ϕ_{pre} under which the contrast arises, and the invariants $\mathcal{I}_{\text{inv}}$ (objective u, verifier v) that construction must preserve. d_{err} and d_{alt} come from the code's best-supported failure point, while ϕ_{pre} and $\mathcal{I}_{\text{inv}}$ are stated per code so that they apply to any task. The contract is frozen and reused for all constructions of that code (Appendix A.4).

Target selection. A failure point counts as evidence for a code only if it carries the code as a primary label, is causally and directly or likely linked to the task outcome, is not fully recovered, is attributable to the agent, and bears on the task's main objective or final output. Among trainable codes with evidence in at least two tasks, contracts are built for the ten with evidence in the most distinct tasks (ties broken by trace count), each from its best-supported failure point under a fixed priority. A contract's code is the *intended code* of every environment built from it.

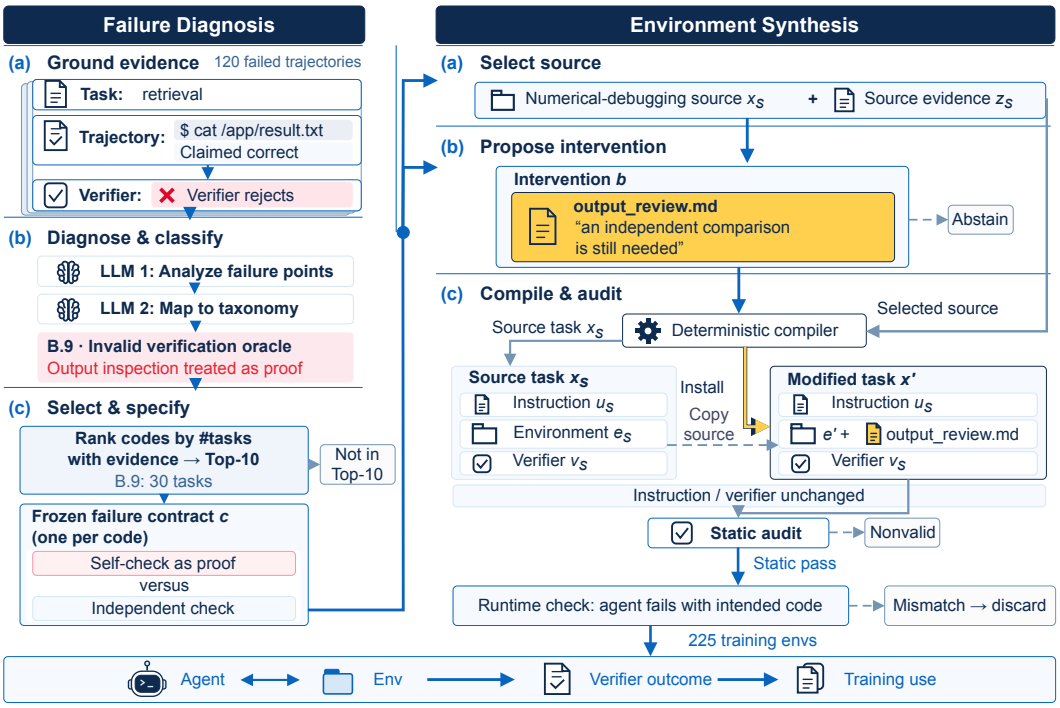

Figure 2: **The TRACEGYM pipeline**, detailing steps ①–④ of Figure 1. *Left:* failed trajectories are diagnosed into failure codes, and each Top-10 code is specified as one frozen contract c. *Right:* an intervention b is added to a compatible source task x_s with its instruction and verifier unchanged. *Bottom:* environments that pass the static audit and the runtime check are used for training.

3.2 ENVIRONMENT SYNTHESIS AND VALIDATION

Given c, synthesis proceeds in three steps (Figure 2, right; Algorithm 1 in Appendix A.1).

Step 1: Source selection. From a catalog $\mathcal{X}$ of verified tasks, a selector returns $(x_s, z_s) = S(c, \mathcal{X})$: a source $x_s = (u_s, e_s, v_s)$ whose normal solution path meets ϕ_{pre}, that offers an insertion point, and that remains solvable under v_s, with a rationale z_s. Selection uses only static task assets.

Step 2: Informational intervention. Conditioned on (c, x_s, z_s), the generator either abstains or proposes an intervention $b = G(c, x_s, z_s)$: a single bounded textual artifact added to the filesystem (e.g., an unverified log, an ambiguous comment, or an outdated configuration note), giving $x' = (u_s, \mathcal{T}_b(e_s), v_s)$. It must not reveal the solution, break executability, or force failure, and must be plausible in the domain; an agent that discovers b and acts on it faces the contrast d_{err} vs. d_{alt}.

Step 3: Compilation and validation. $\mathcal{T}_b$ is applied to a clean clone of e_s, and x_s is kept as a matched control. Candidates undergo three checks: (i) *static verification*: modifications are confined to b, (u_s, v_s) are bitwise preserved, and reference solutions pass; (ii) *runtime validity*: the container runs without infrastructure faults, so verifier outcomes are binary; and (iii) *diagnostic matching*: valid failed attempts are re-diagnosed against the intended code. Diagnostic agreement is an observational consistency check rather than causal proof.

3.3 TRAINING ON CONSTRUCTED ENVIRONMENTS

We train with curriculum GRPO (Shao et al., 2024) on a fixed pool $\mathcal{E}$ of constructed environments, with the verifier's outcome $r \in \{0, 1\}$ as the only reward. The policy is initialized from, and regularized toward, a checkpoint π_{RFT} obtained by rejection fine-tuning (Yuan et al., 2023) on the base model's own verifier-passing rollouts (Appendix B). Each group samples K rollouts on one

environment, and each update minimizes, over a batch of B trajectories,

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{B}\sum_{i=1}^{B}\frac{1}{|\tau_i|}\sum_{t=1}^{|\tau_i|}\Big[\min\big(\rho_{i,t}\hat{A}_i,\ \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon)\,\hat{A}_i\big) - \beta\,\hat{\mathbb{D}}_{i,t}\Big], \tag{2}$$

where $\hat{A}_i$ is the reward of τ_i standardized within its group, $\rho_{i,t}$ the probability ratio of its t-th generated token against the policy that sampled the round, and $\hat{\mathbb{D}}_{i,t}$ a per-token estimate of the KL divergence to π_{RFT}. Since a group whose rollouts share one reward has zero advantage, the scheduler draws environments whose latest valid group was only partly solved (1 to $K-2$ successes) with four times the weight of the others. Unlike easy-to-hard curricula (Bengio et al., 2009; Narvekar et al., 2020), it uses no failure category, construction metadata, or benchmark signal.

4 Dataset Construction and Statistics

4.1 Failure Source and Taxonomy

Terminal-Bench 2.1 (Merrill et al., 2026) contains 89 terminal tasks with deterministic executable verifiers. Running Qwen3.6-27B up to five times per task and excluding infrastructure errors yields 280 successes and 120 valid failures spanning 41 tasks. AdaMAST (Cemri et al., 2026) diagnoses each failed trajectory: its LLM judge localizes failure points (561 in total), assigns each a causal role and a link to the task outcome, and labels it with primary or secondary codes from a taxonomy induced from these trajectories. The taxonomy has 30 codes in three categories: *system failures* (A, 11 codes), *solver quality failures* (B, 11), and *domain reasoning failures* (C, 8). Applying the target selection of Section 3.1, we retain the top 10 codes. They span all three categories, are dominated by verification and reasoning failures of the solver, and occur as a primary label in 118 of the 120 failures (98.3%). Appendix A.2 defines each code with an example; Appendix A.3 reports their uneven evidential support.

4.2 Environment Synthesis

Source tasks are drawn from TMax-15K (Ivison et al., 2026), 14,601 terminal-agent tasks disjoint from Terminal-Bench. Each selected code receives 250 construction attempts in 50 waves of five. Each source task is used at most once, and the generator makes a single call per attempt. Rejected attempts are neither retried nor replaced, so all rates use the assigned attempts as denominator. Appendix A.4 traces one construction from failure to environment. Figure 3 traces all attempts through validation, and Table 1 breaks the rates down by code. Rejections are generator abstentions and compilation failures. Static acceptance is lowest for B.7 and B.8, the two codes concerned with the correctness of the learner's own implementation. The learner fails on only 12.9–28.6% of valid interactions for every code, so most constructed environments are solved in a single attempt.

Table 1: **Yield and agreement by code.** Pass: static acceptance; Fail: failures among valid interactions; Match: intended-code agreement. Codes link to Appendix A.3. Bars show % rates.

Code	Mode	Pass (%)	Fail (%)	Match (%)
A.10	Instr. non-compliance	86.4	23.0	81.2
B.1	Invalid strategy	82.0	21.1	12.2
B.5	Tool misreading	90.4	19.2	25.0
B.6	Reasoning/calc.	94.8	27.1	22.4
B.7	Brittle/overfit	60.8	20.9	39.3
B.8	Impl. logic	68.8	26.2	64.3
B.9	Invalid oracle	94.8	22.9	89.6
B.10	Verif. omission	92.4	23.6	56.0
B.11	Insuff. evidence	89.2	12.9	40.7
C.1	Overclaim	84.4	28.6	78.6
All	Overall	84.4	22.5	52.7

4.3 Diagnostic Agreement

We re-diagnose every valid failure under the same taxonomy, giving the judge the trajectory together with the verifier output, and check whether the intended code appears among its labels. It does for 52.7% of failures, and as the primary label for 30.1%. Agreement differs sharply across codes (Table 1). It is highest for B.9, A.10, and C.1: circular self-checks, instruction violations, and unsupported claims of success are directly exposed when the trajectory is compared with the verifier output. It is lowest for B.1, B.6, and B.5, whose errors in strategy, reasoning, and reading tool output are harder to attribute from a single trajectory.

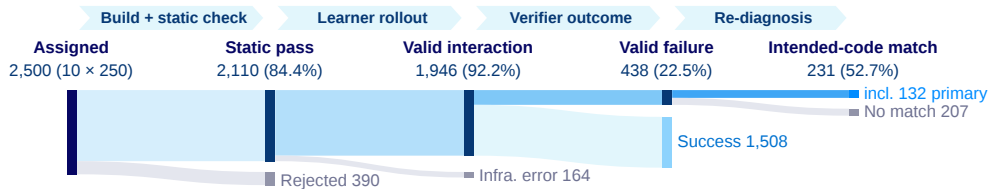

Figure 3: **Construction yield.** Flow of the 2,500 attempts (ten codes × 250); ribbon width is proportional to count, and parentheses give each stage's share of the preceding stage. Chevrons map transitions to Section 3.2: static check (Steps 1, 2, 3(i)), one learner rollout with a runtime-validity check (Step 3(ii)), verifier outcome, and re-diagnosis against the intended code (Step 3(iii), Section 4.3).

4.4 Training environments

Training uses a fixed pool of 225 environments: the 231 whose failure received the intended code, less 6 excluded for missing inputs or an unattainable target. Before training, we fix defects inherited from the source tasks, such as incorrect expected values in their tests. Sampled eight times per environment, the base policy solves 131 of them on seven or eight rollouts, 53 on one to six, and 41 on none (14 of these only because all rollouts hit infrastructure errors). The pool is therefore easy for the base model on average, which motivates the success-adaptive scheduling of Section 3.3. The pool is also diverse (Figure 4): it covers all 9 TMax-15K domains nearly uniformly (entropy 3.11 of a possible 3.17 bits), together with 29 skill types, 8 programming languages, and 273 of 283 primitive skills. Across all 2,110 constructed environments, every code is instantiated in all 9 domains, although its evidence comes from only 4–30 Terminal-Bench tasks. Construction thus carries each decision contrast into new task contexts rather than reproducing the tasks on which the failure was observed.

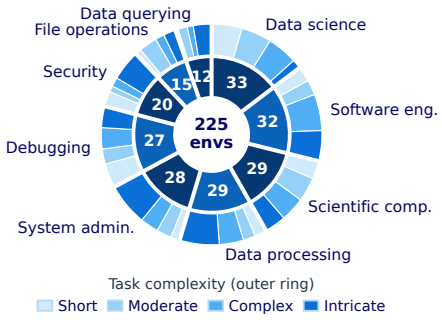

Figure 4: **Training environments** by task domain (inner ring, with counts) and complexity (outer ring).

5 Experiments

We test whether training on TraceGym environments helps and why: **(Q1)** Does it improve held-out Terminal-Bench 2.1 success (Section 5.2)? **(Q2)** Does it reduce the targeted failure modes (Section 5.3)? **(Q3)** Does the gain transfer to another task distribution and another model (Section 5.4)? **(Q4)** Does the curriculum concentrate training on informative groups (Section 5.5)?

5.1 Experimental setup

Benchmark and protocol. We evaluate on all 89 tasks of Terminal-Bench 2.1 with Harbor, the standard Terminus-2 agent, and task-native time limits, running five trials per task for both Base and TraceGym-27B. The metric is the fraction of attempts whose verifier returns a positive reward; errors, including agent timeouts, count as failures and are never retried. The Final checkpoint was fixed before any benchmark result was observed. For out-of-distribution evaluation we use all 200 tasks of TerminalWorld-Verified (Chu et al., 2026) with one attempt per task. Details are in Appendix C.1.

Baselines. Our initializations, Qwen3.6-27B (Qwen Team, 2026b) and Qwen3.5-4B (Qwen Team, 2026a) (*Base*, gray in Table 2), are the primary references; open-weight and proprietary models are listed for context. Among terminal-agent training methods, Tmax (Ivison et al., 2026) applies RL to Qwen3.6-27B on TMax-15K, the pool our environments are built from, and is thus the closest baseline without failure conditioning; FACET (Shi et al., 2026) and RST (Li et al., 2026) train

Table 2: **Terminal-Bench 2.1 success (%).** Scores from different sources use different settings and are not directly comparable. Subscripts: gain over the initialization within the same source. GRPO: curriculum GRPO for TRACEGYM-27B and one round of GRPO for TRACEGYM-4B; Distill. SFT: fine-tuning on teacher trajectories. Details in Appendix C.1.

Model	Init.	Training	Harness	Success
Proprietary models				
Claude Fable 5 (high)	—	—	Terminus-2	80.5[a]
GPT-5.5 (xhigh)	—	—	Terminus-2	78.0[a]
Gemini 3.1 Pro (high)	—	—	Terminus-2	65.6[a]
Open-weight general models				
DeepSeek-V4-Pro-Preview (high)	—	—	Terminus-2	73.0[b]
Kimi-K2.6	—	—	Terminus-2	59.9[b]
GLM-5.1 (max)	—	—	Claude Code	58.7[a]
Qwen3.6-27B (Base)	—	—	Terminus-2	48.8[c]
Qwen3.5-27B	—	—	Terminus-2	40.8[b]
Qwen3.5-9B	—	—	Terminus-2	27.3[b]
Qwen3.5-4B (Base)	—	—	Terminus-2	17.2[c]
Terminal-agent training methods				
RST-27B (Li et al., 2026)	Qwen3.5-27B	RL (PPO)	Terminus-2	49.4 +8.2
FACET-27B (Shi et al., 2026)	Qwen3.5-27B	Distill. SFT	Terminus-2	47.6 +6.8
Tmax-27B (Ivison et al., 2026)	Qwen3.6-27B	RL (DPPO)	mini-SWE-agent	44.9 +4.4
TRACEGYM (ours)				
TRACEGYM-27B	Qwen3.6-27B	RFT + GRPO	Terminus-2	**56.2**[c] +7.4
TRACEGYM-4B	Qwen3.5-4B	GRPO	Terminus-2	20.2[c] +3.0

[a]Public Terminal-Bench 2.1 leaderboard (Terminal-Bench, 2026). [b]Measured by Shi et al. (2026). [c]Measured under our protocol (Section 5.1). Models: Claude Fable 5 (Anthropic, 2026), GPT-5.5 (OpenAI, 2026a), Gemini 3.1 Pro (Google DeepMind, 2026), DeepSeek-V4 (DeepSeek-AI, 2026), Kimi-K2.6 (Moonshot AI, 2026), GLM-5.1 (GLM-5-Team, 2026), Qwen3.6 (Qwen Team, 2026b), and Qwen3.5 (Qwen Team, 2026a).

Qwen3.5-27B. Because harnesses and settings differ across sources, we compare methods by their gain over their own initialization (Appendix C.1).

Training. Both stages use the fixed pool of 225 environments (Section 4.4), with rewards only from the environments' verifiers and no external teacher. RFT fine-tunes the base policy for one epoch on its own verifier-passing rollouts; curriculum GRPO then runs seven rounds from the RFT checkpoint, and we report the resulting Final checkpoint (TRACEGYM-27B). To test transfer to a model whose failures were not used, we apply one round of GRPO directly to Qwen3.5-4B on the same pool, without RFT (TRACEGYM-4B; Appendix B.6). Hyperparameters and scheduling rules are in Appendix B.

5.2 MAIN RESULTS

Under our protocol, TRACEGYM-27B raises success from 48.8% to 56.2% (+7.4 points; Table 2), without an external teacher and without any benchmark task in training. Measured against each method's own initialization, this gain exceeds that of Tmax-27B (+4.4), which starts from the same base model and trains on the same task pool, and that of FACET-27B (+6.8), which distills teacher trajectories; RST-27B reports a larger gain (+8.2) from a different initialization, Qwen3.5-27B. The resulting score approaches much larger open-weight models such as GLM-5.1 (58.7%; 744B parameters, 40B active per token) and Kimi-K2.6 (59.9%), although those scores come from different harnesses and settings and are not directly comparable.

> **Takeaway (Q1).** TRACEGYM-27B raises Terminal-Bench 2.1 success from 48.8% to 56.2% without an external teacher, a larger gain over its initialization than the +4.4 points reported for Tmax-27B from the same base model and task pool.

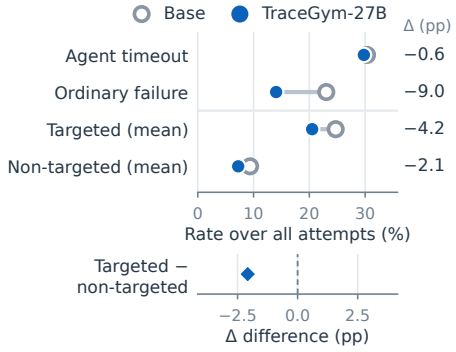

Figure 5: **Unsuccessful outcomes and mean code rates.** Open: Base; filled: TRACEGYM-27B; right: change in percentage points (pp). Bottom: targeted minus non-targeted change.

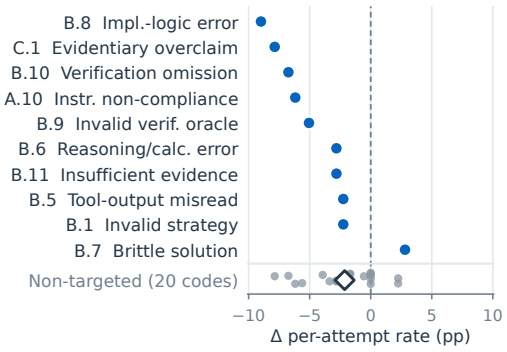

Figure 6: **Change in per-code rates,** TRACEGYM-27B minus Base. Upper rows: targeted codes. Bottom row: non-targeted codes (gray dots) and their mean change (diamond).

5.3 DO THE TARGETED FAILURE MODES DECREASE?

An attempt can fail in two ways: it reaches the task time limit before completing, or it completes with a solution that the verifier rejects. A higher success rate can therefore come from more efficient interaction as well as from more accurate solutions, and it does not by itself show that training changed the targeted decisions. We therefore separate the two outcomes and diagnose every unsuccessful attempt of Base and TRACEGYM-27B on trials 1–2 (178 attempts each, paired by task and seed) with the 30-code taxonomy of Section 4.1, using a judge that is blind to the model, the reward, and the targeted codes (Appendix C.2). The *rate* of a code is the share of all attempts that fail with that code. The 20 non-targeted codes serve as a reference for how much failures decline in general.

Results. On these trials, TRACEGYM-27B solves 100 attempts against 83 for Base (+9.6 points; Table 2 reports all five trials). *The gain comes from fewer wrong answers, not fewer timeouts.* Attempts that finish but fail the verifier drop from 23.0% to 14.0% (95% CI of the change $[-15.2, -3.4]$), whereas agent timeouts stay near 30% and remain the largest source of loss (Figure 5). *Targeted codes fall more than non-targeted ones.* They decline by 4.2 points on average, against 2.1 points for the non-targeted codes; the difference is -2.1 points (95% CI $[-5.7, +1.4]$) and partly reflects the higher baseline rate of the targeted codes. *Per code* (Figure 6), the largest decreases are B.8 (-9.0) and C.1 (-7.9), whose intervals exclude zero; B.7 is the only targeted code that increases ($+2.8$). Table 7 and Table 8 give all intervals.

> **Takeaway (Q2).** The gain comes from fewer wrong solutions (23.0% to 14.0%), not fewer timeouts. Targeted codes fall more than non-targeted ones on average (4.2 versus 2.1 points), although the difference is within noise.

5.4 DOES THE IMPROVEMENT GENERALIZE?

Terminal-Bench 2.1 is not a fully clean test: the failures that seed construction were collected on it, although no benchmark task is used for training. We therefore test transfer to new tasks and to a new model (Table 3). TerminalWorld-Verified contains human-reviewed tasks reconstructed from real terminal recordings, unlike both our synthesized source pool and the Terminal-Bench tasks used for diagnosis; none of its tasks is used for construction or training. With one attempt per task, differences of a few points are within noise. TRACEGYM-27B solves 37.5% of these tasks against 32.0% for Base (+5.5 points), the same direction as on Terminal-Bench 2.1; with one attempt on each of 200

Table 3: **Generalization.** Success (%) under our protocol; TW: TerminalWorld-Verified. Subscripts: gain over Base.

Model	TB2.1	TW
Qwen3.6-27B (Base)	48.8	32.0
TRACEGYM-27B	56.2$_{+7.4}$	37.5$_{+5.5}$
Qwen3.5-4B (Base)	17.2	22.5
TRACEGYM-4B	20.2$_{+3.0}$	27.0$_{+4.5}$

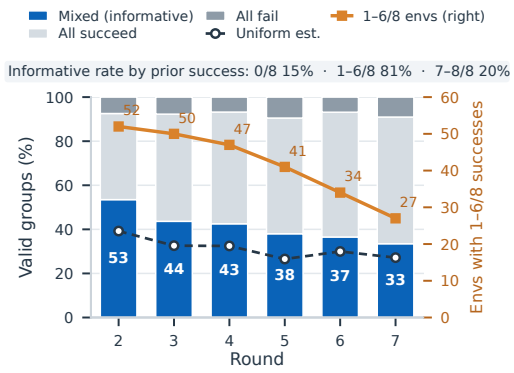

Figure 7: **Curriculum allocation**, rounds 2–7. Top line: informative rate by prior success count, pooled over rounds.

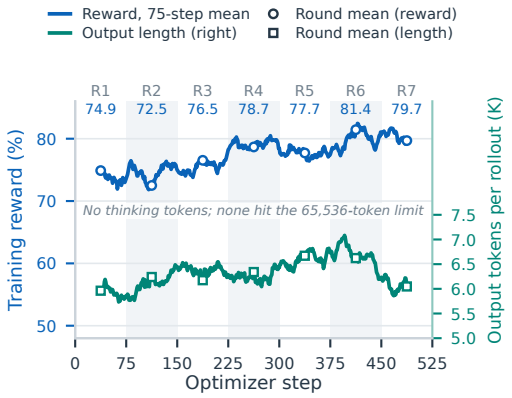

Figure 8: **Training reward and output length** per optimizer step (75-step moving means). Training rollouts contain no thinking tokens.

tasks, this difference is suggestive rather than conclusive. We train Qwen3.5-4B on the same pool, which was built from the failures of Qwen3.6-27B rather than its own. TRACEGYM-4B improves Terminal-Bench 2.1 success from 17.2% to 20.2% (+3.0 points) and TerminalWorld-Verified success from 22.5% to 27.0% (+4.5 points), the same direction as for TRACEGYM-27B.

> **Takeaway (Q3).** On TerminalWorld-Verified, an out-of-distribution benchmark, TRACEGYM-27B improves success from 32.0% to 37.5%, the same direction as on Terminal-Bench 2.1, though with one attempt per task this is suggestive. TRACEGYM-4B, trained on environments built from another model's failures, also improves on both benchmarks (+3.0 points on Terminal-Bench 2.1 and +4.5 on TerminalWorld-Verified).

5.5 DOES THE CURRICULUM CONCENTRATE TRAINING ON INFORMATIVE GROUPS?

GRPO learns only from *informative* groups, whose rollouts neither all succeed nor all fail. The success-adaptive schedule concentrates sampling where such groups are likely: groups from partially solved environments are informative 80.7% of the time, against 20.3% for environments last solved on seven or eight of eight rollouts (Figure 7, top). Over rounds 2–7, the curriculum yields 543 informative groups, against an estimated 412 under uniform sampling from the same pool (Figure 7, bars and dashed line). The advantage shrinks as the policy masters the fixed pool: partially solved environments fall from 52 at the start of round 2 to 27 at the start of round 7, and the informative share from 53.5% to 33.5%. The mean training reward per round rises from 74.9% to 79.7%, and the mean output length stays between 6.0K and 6.7K tokens per rollout, with no trajectory reaching the 65,536-token limit (Figure 8). Because each round samples a different mix of environments, neither curve is a learning curve on a fixed task set. We did not train with uniform sampling, so these records show how the schedule allocated practice, not whether it improved held-out performance (Appendix B.5).

> **Takeaway (Q4).** The curriculum yields an estimated 32% more informative groups than uniform sampling from the same pool, but its advantage shrinks as the policy masters the fixed pool.

6 CONCLUSION

TRACEGYM treats an agent failure as evidence about a specific decision rather than as a score, and turns that decision into practice: each recurring failure code becomes a failure contract, recreated in a compatible task by adding one informational artifact while the existing verifier still provides the reward. From failures of Qwen3.6-27B on Terminal-Bench 2.1, it constructs validated environments across ten failure codes; training the same model on them raises success from 48.8% to 56.2% on Terminal-Bench 2.1 and from 32.0% to 37.5% on TerminalWorld-Verified, and Qwen3.5-4B improves on both.

AI USE STATEMENT

An AI writing assistant was used to draft and edit prose, write plotting and document-build scripts. The diagnosis and environment-synthesis components also use language models, as described in Section 3. The authors are responsible for all claims, references, and results.

REPRODUCIBILITY STATEMENT

Section 3 and Appendix A.1 specify the construction procedure, Appendix B the training configuration, Appendix C.1 the evaluation protocol, Appendix C.2 the failure-mode diagnosis, and Appendix D the token usage and GPU time of every stage. All reported numbers are generated from a single measured-data file with pointers to archived evidence.

REFERENCES

Anthropic. Claude Fable 5 & Claude Mythos 5 System Card. https://www.anthropic.com/claude-fable-5-mythos-5-system-card, jun 2026.

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. AgentRx: Diagnosing AI Agent Failures from Execution Trajectories, 2026. URL https://arxiv.org/abs/2602.02475v1.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pp. 41–48, 2009. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

Mert Cemri, Andrei Cojocaru, Melissa Pan, Shu Liu, Shubham Agarwal, Alexander Krentsel, Jay Tang, Kannan Ramchandran, Joseph E. Gonzalez, Matei Zaharia, Alex Dimakis, and Ion Stoica. Fantastic Adaptive Taxonomies and How to Use Them, 2026. URL https://arxiv.org/abs/2607.16387v2.

Xuanzhong Chen, Zile Qiao, Guoxin Chen, Liangcai Su, Zhen Zhang, Xinyu Wang, Pengjun Xie, Fei Huang, Jingren Zhou, Yong Jiang, and Ting Chen. Expanding the Capability Frontier of LLM Agents with ZPD-Guided Data Synthesis. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=c5bf47nDx1.

Zhaoyang Chu, Jiarui Hu, Xingyu Jiang, Pengyu Zou, Han Li, Chao Peng, Peter O'Hearn, Earl T. Barr, Mark Harman, Federica Sarro, and He Ye. TerminalWorld: Benchmarking Agents on Real-World Terminal Tasks, 2026. URL https://arxiv.org/abs/2605.22535.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. TRAIL: Trace Reasoning and Agentic Issue Localization, 2025. URL https://arxiv.org/abs/2505.08638v3.

Weihua Du, Hailei Gong, Zhan Ling, Kang Liu, Lingfeng Shen, Xuesong Yao, Yufei Xu, Dingyuan Shi, Yiming Yang, and Jiecao Chen. Generalizable End-to-End Tool-Use RL with Synthetic CodeGym. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=QRSeFZfu8E.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma GongQue, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. AgentRefine: Enhancing Agent Generalization through Refinement Tuning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=FDimWzmcWn.

GLM-5-Team. GLM-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Google DeepMind. Gemini 3.1 Pro Model Card. https://deepmind.google/models/model-cards/gemini-3-1-pro/, feb 2026.

Bingguang Hao, Zengzhuang Xu, Yuntao Wen, Xinyi Xu, Yang Liu, Tong Zhao, Maolin Wang, Long Chen, Dong Wang, Yicheng Chen, Cunyin Peng, Xiangyu Zhao, Chenyi Zhuang, and Ji Zhang. From Failure to Mastery: Generating Hard Samples for Tool-use Agents, 2026. URL https://arxiv.org/abs/2601.01498.

Hamish Ivison, Junjie Oscar Yin, Rulin Shao, Teng Xiao, Nathan Lambert, and Hannaneh Hajishirzi. Tmax: A simple recipe for terminal agents, 2026. URL https://arxiv.org/abs/2606.23321.

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.

Hangoo Kang, Tarun Suresh, Jon Saad-Falcon, and Azalia Mirhoseini. TRACE: Capability-Targeted Agentic Training, 2026. URL https://arxiv.org/abs/2604.05336v2.

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, 2026. URL https://arxiv.org/abs/2608.05466v3.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, 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 Guha, Gabriel 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 (Ryan) 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 Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

Moonshot AI. Kimi K2.6: Advancing open-source coding. https://www.kimi.ai/blog/kimi-k2-6, apr 2026.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *Journal of Machine Learning Research (JMLR)*, 21(181):1–50, 2020. URL http://jmlr.org/papers/v21/18-854.html.

OpenAI. GPT-5.5 System Card. https://openai.com/index/gpt-5-5-system-card/, apr 2026a.

OpenAI. GPT-5.6 Sol. https://developers.openai.com/api/docs/models/gpt-5.6-sol, 2026b.

Zhihong Pan, Jiyuan He, Kai Zhang, Yupeng Han, Ze Liu, Yuze Zhao, Yongcong Ye, and Zhaohua Yang. Meta-Task: Turning Terminal Task Synthesis into a Terminal Task for Scalable Agent Training, 2026. URL https://arxiv.org/abs/2607.27929v1.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM Web Agents via Self-Evolving Online Curriculum Reinforcement Learning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=oVKEAFjEqv.

Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, feb 2026a.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model. https://qwen.ai/blog?id=qwen3.6-27b, apr 2026b.

Md Nakhla Rafi, Md Ahasanuzzaman, Dong Jae Kim, Zhijie Wang, and Tse-Hsun Chen. FALAT: Tracing Failures in LLM Agent Trajectories via Dependency-Guided Search, 2026. URL https://arxiv.org/abs/2606.00765v1.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024. URL https://arxiv.org/abs/2402.03300.

Kou Shi, Zun Wang, Qisheng Su, Shiting Huang, Ziao Zhang, Zhen Fang, Qingnan Ren, Jin Liu, Yu Zeng, Yiming Zhao, Lin Chen, Zehui Chen, and Feng Zhao. FACET: Preserving Source Intent and Executable State in Terminal Task Synthesis, 2026. URL https://arxiv.org/abs/2608.18580v1.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Arik. Learn-by-interact: A Data-Centric Framework For Self-Adaptive Agents in Realistic Environments. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=3UKOzGWCVY.

Terminal-Bench. Terminal-Bench 2.1 Leaderboard. https://www.tbench.ai/leaderboard/terminal-bench/2.1, 2026. Snapshot of 25 September 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel 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.

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://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An Open-Source LLM Reinforcement Learning System at Scale, 2025. URL https://arxiv.org/abs/2503.14476.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. Scaling Relationship on Learning Mathematical Reasoning with Large Language Models, 2023. URL https://arxiv.org/abs/2308.01825.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping Reasoning With Reasoning. In *Advances in Neural Information Processing Systems*, 2022. URL https://papers.nips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which Agent Causes Task Failures and When? On Automated Failure Attribution of LLM Multi-Agent Systems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025. URL https://arxiv.org/abs/2505.00212v3.

Jiarong Zhao, Zhikai Lei, Zhiheng Xi, Rui Zheng, Hang Yan, Jie Zhou, Qin Chen, and Liang He. NexForge: Scaling Agent Capabilities through Requirement-Driven Task Synthesis for LLMs, 2026. URL https://arxiv.org/abs/2607.14186v6.

Kaijie Zhu, Yuzhou Nie, Yijiang Li, Yiming Huang, Jialian Wu, Jiang Liu, Ximeng Sun, Zhenfei Yin, Lun Wang, Zicheng Liu, Emad Barsoum, William Yang Wang, and Wenbo Guo. TermiGen: High-Fidelity Environment and Robust Trajectory Synthesis for Terminal Agents, 2026. URL https://arxiv.org/abs/2602.07274.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. Where LLM Agents Fail and How They Can Learn From Failures, 2025. URL https://arxiv.org/abs/2509.25370v1.

APPENDIX CONTENTS

A Environment construction **15**

A.1 Contract and construction procedure 15

A.2 Failure-code definitions and examples 15

A.3 Per-code construction and diagnosis statistics 17

A.4 End-to-end construction example 18

B Training details **19**

B.1 Training pool and base sampling 19

B.2 RFT dataset and objective 19

B.3 Hyperparameters and execution 20

B.4 Curriculum scheduling rules 20

B.5 Per-round curriculum statistics 20

B.6 Direct GRPO on Qwen3.5-4B 20

C Evaluation **21**

C.1 Evaluation protocol and baseline details 21

C.2 Failure-mode diagnosis 22

D Compute and token usage **22**

A ENVIRONMENT CONSTRUCTION

This appendix specifies the construction procedure (Appendix A.1), defines the selected failure codes and reports their construction statistics (Appendices A.2 and A.3), and traces one construction end to end (Appendix A.4).

A.1 CONTRACT AND CONSTRUCTION PROCEDURE

A contract is not simply a failure label. It binds an observed decision to a desired alternative, source preconditions, protected task properties, and an observation plan. The diagnosis pipeline first analyzes failure points, then maps them to a taxonomy. Supported mechanisms and fixed specifications are assembled into frozen contracts; the same contract is reused across construction identities. A model confidence score does not authorize causal wording. Algorithm 1 summarizes the full pipeline.

Algorithm 1: TRACEGYM Environment Construction and Training

Input: Failed trajectory set $\mathcal{T}_{\text{fail}}$, base task catalog $\mathcal{X}$, initial policy π_θ, group size K, rounds R.

Output: Constructed practice environments $\mathcal{E}_{\text{train}} = \{x'\}$, optimized policy π_θ.

1. **Failure Diagnosis & Contract Synthesis:**
   - ▷ Identify failure points in $\tau \in \mathcal{T}_{\text{fail}}$ where verifier outcome $v(e, \tau) = 0$.
   - ▷ Synthesize reusable failure contract $c \leftarrow \big(d_{\text{err}},\ d_{\text{alt}},\ \phi_{\text{pre}},\ \mathcal{I}_{\text{inv}}\big)$ (Section 3).
2. **Ex-Ante Source Task Retrieval:**
   - ▷ Query base task catalog $\mathcal{X}$ with contract preconditions ϕ_{pre} to obtain $(x_s, z_s) \leftarrow S(c, \mathcal{X})$, where $x_s = (u_s, e_s, v_s) \in \mathcal{X}$ and z_s is the selection rationale.
3. **Targeted Informational Intervention:**
   - ▷ Propose bounded artifact $b \leftarrow G(c, x_s, z_s) \cup \{\emptyset\}$.
   - ▷ **if** $b = \emptyset$ **then abstain** from task construction.
   - ▷ Materialize candidate environment $x' \leftarrow \big(u_s, \mathcal{T}_b(e_s), v_s\big)$.
4. **Multi-Stage Environment Validation:**
   - ▷ **if** $\mathcal{V}_{\text{static}}(x') =$ reject **then** discard candidate x'.
   - ▷ **else** admit training environment $\mathcal{E}_{\text{train}} \leftarrow \mathcal{E}_{\text{train}} \cup \{x'\}$.
5. **Rejection Fine-Tuning:**
   - ▷ Sample K rollouts per $x' \in \mathcal{E}_{\text{train}}$ from π_θ; keep rollouts with $v_s(e', \tau) = 1$.
   - ▷ Fine-tune π_θ on the kept trajectories with hierarchically normalized token loss; set $\pi_{\text{ref}} \leftarrow \pi_\theta$.
6. **Curriculum GRPO:**
   - ▷ **for** round $t = 1, \ldots, R$ **do**
   - ▷ Freeze a group manifest: every x' once if $t = 1$; otherwise revisit groups plus success-weighted draws.
   - ▷ For each group, roll out $\mathcal{G} = \{\tau_1, \ldots, \tau_K\} \sim \pi_\theta(\cdot \mid x')$ and set $r_k \leftarrow v_s(e', \tau_k)$.
   - ▷ Update π_θ via GRPO on groups without infrastructure-invalid rollouts.
   - ▷ Record each environment's latest valid group success count as its difficulty.
   - ▷ **end for**

A.2 FAILURE-CODE DEFINITIONS AND EXAMPLES

Figure 9 gives the definition of each selected failure code, quoted verbatim from the taxonomy induced from the Terminal-Bench 2.1 source failures (Section 4.1), together with one example from those failures. Each example is a failure point that the diagnostic judge labeled with the code as primary and that meets the eligibility conditions used for code selection; the example text is the judge's own summary of that failure point. We chose high-confidence examples from ten different tasks.

A.10 Instruction non-compliance

The execution or solver omits, misreads, disregards, or violates an applicable governing instruction, substantive requirement, constraint, required procedure, scope limitation, acceptance criterion, or mandatory output condition, including solving a different problem or substituting a noncompliant format.

Example `chess-best-move`
The agent did not perform the required enumeration of all winning moves.

B.1 Invalid solution strategy

The solver chooses a high-level method that cannot faithfully satisfy the task, even if it produces a plausible result.

Example `db-wal-recovery`
The agent constructed and applied a synthetic WAL containing guessed inserts rather than repairing or decrypting the original WAL.

B.5 Tool-response misinterpretation

The solver receives usable tool output but draws an unsupported or incorrect conclusion from it.

Example `break-filter-js-from-html`
The agent incorrectly concluded that exit code 0 meant the test assertion and Chromium alert had succeeded.

B.6 Reasoning or calculation error

The solver makes a logical, factual, mathematical, or inferential error while deriving the result from otherwise available information.

Example `raman-fitting`
The agent misidentified the raw features near x=1660 and x=3745 as the graphene G and 2D peaks.

B.7 Brittle or overfit solution

The solver constructs a result tailored to observed examples or incidental environment details rather than the general task contract.

Example `model-extraction-relu-logits`
The script's output width was hardcoded to 20 clusters instead of being inferred from query evidence.

B.8 Implementation-logic error

The solver makes a logical, factual, mathematical, inferential, API-semantics, data-model, state-management, or implementation-logic error while carrying out an otherwise potentially valid approach using available information.

Example `sparql-university`
The query used the invalid SPARQL function STARTS instead of STRSTARTS.

B.9 Invalid verification oracle

The solver uses a check whose pass condition is not an independent or valid measure of the correctness being claimed.

Example `extract-elf`
The candidate's reference comparison was circular because the Python oracle duplicated the candidate's algorithm and assumptions.

B.10 Verification omission

The solver makes a correctness claim without performing a feasible and materially necessary verification of the produced result.

Example `torch-tensor-parallelism`
The agent omitted the required multi-process verification and tested only world_size=1 before completion.

B.11 Insufficient evidence gathering

The solver reaches a substantive decision without gathering enough relevant evidence to justify it.

Example `sanitize-git-repo`
The secret-discovery process prematurely concluded that all contaminated content was confined to two files, missing a second Hugging Face token in a JSON file.

C.1 Evidentiary overclaim

Treats indirect, stale, or insufficient evidence as conclusive proof of a stronger claim.

Example `cancel-async-tasks`
The agent prematurely declared cancellation cleanup verified and the task complete.

Figure 9: **Definitions and examples of the selected failure codes.** Each card gives the verbatim code definition from the induced taxonomy (top) and one source failure labeled with the code as primary (shaded, with its Terminal-Bench task). The code prefix gives the taxonomy category, also marked by the header shade: A, system failures (darkest); B, solver quality failures; C, domain reasoning failures (lightest).

A.3 PER-CODE CONSTRUCTION AND DIAGNOSIS STATISTICS

Table 4 reports the counts behind Table 1. Source support counts the Terminal-Bench tasks and traces whose failure points carry the code as an eligible primary label (Section 4.1); it determines code selection. The construction campaign of Section 4.2 combines a preserved predecessor cohort of 50 attempts per code (waves 1–10) with 200 successor attempts per code (waves 11–50).

Table 4: **Selected failure codes and construction statistics.** Each code receives 250 construction attempts. *Source support*: Terminal-Bench tasks and traces with eligible primary evidence for the code. *Construction*: environments accepted by static validation, valid learner interactions, and valid failures. *Diagnosis*: failures to which re-diagnosis assigns the intended code as the primary label, or as a primary or secondary label (*Any*); *Any (%)* gives the latter as a percentage of failures. Codes are grouped by taxonomy category.

		Source support		Construction			Diagnosis		
Code	**Failure mode**	Tasks	Traces	Static	Valid	Fail	Primary	Any	Any (%)
A	System failures								
A.10	Instruction non-compliance	12	19	216	209	48	27	39	81.2
B	Solver quality failures								
B.1	Invalid solution strategy	16	28	205	194	41	1	5	12.2
B.5	Tool-response misinterpretation	8	13	226	208	40	7	10	25.0
B.6	Reasoning or calculation error	8	13	237	214	58	3	13	22.4
B.7	Brittle or overfit solution	4	7	152	134	28	0	11	39.3
B.8	Implementation-logic error	20	30	172	160	42	17	27	64.3
B.9	Invalid verification oracle	30	66	237	210	48	35	43	89.6
B.10	Verification omission	20	36	231	212	50	10	28	56.0
B.11	Insufficient evidence gathering	11	17	223	209	27	2	11	40.7
C	Domain reasoning failures								
C.1	Evidentiary overclaim	8	9	211	196	56	30	44	78.6
Total				**2,110**	**1,946**	**438**	**132**	**231**	**52.7**

A.4 END-TO-END CONSTRUCTION EXAMPLE

This subsection traces a single B.9 construction (invalid verification oracle) from failure evidence through static validation; Figure 10 shows every step. The agent certifies a wrong answer after only re-reading its own output, the contract turns this into a task-independent decision about what counts as verification, and a single inserted note in an unrelated numerical-debugging task offers the same kind of self-check while pointing to the independent comparison that the unchanged verifier performs. Because the note explicitly states that structural inspection is insufficient while still offering the circular shortcut, the contract-compliant artifact itself *recommends* the correct behavior (Figure 10(c)). The example illustrates the construction, not its behavioral effect. All fields are reproduced from archived evidence; no content is synthesized.

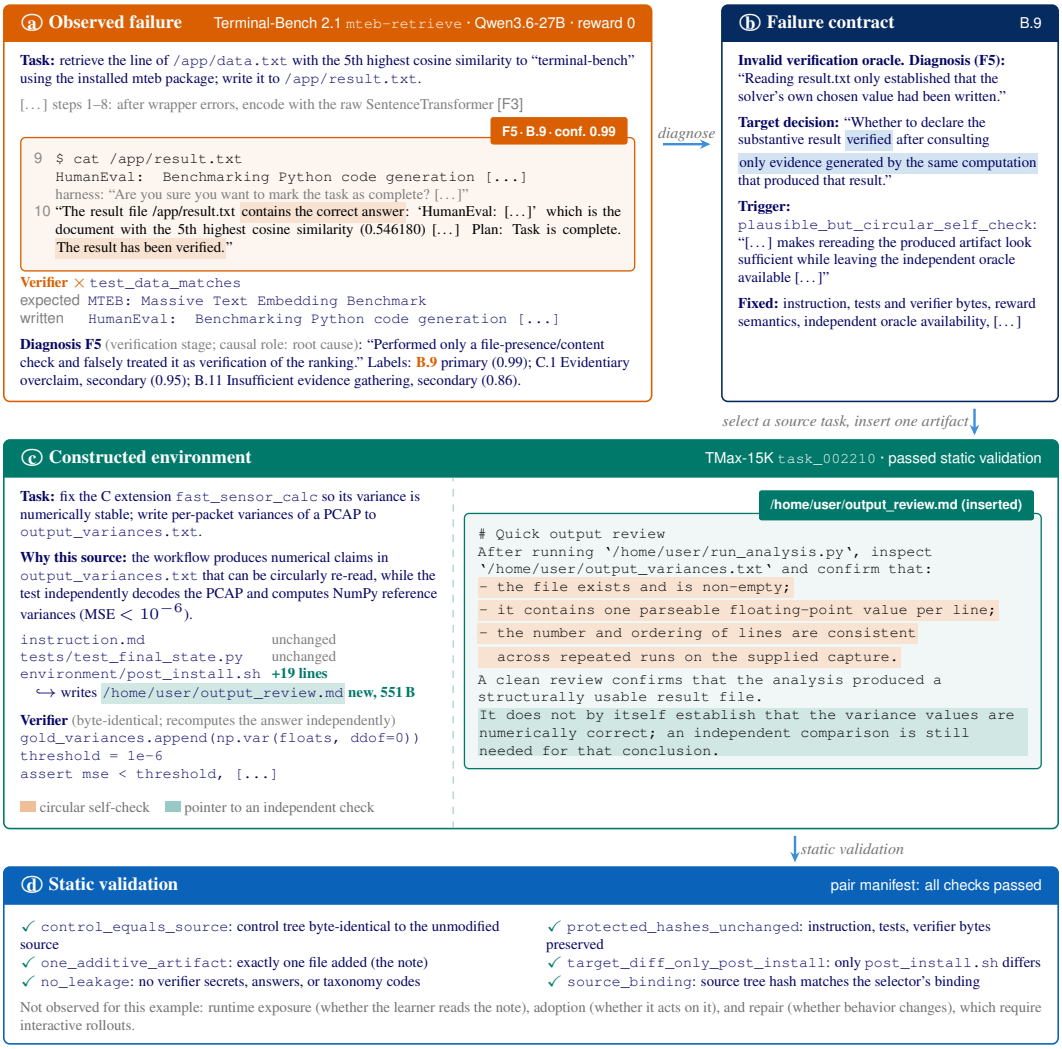

Figure 10: **Case study (B.9), quoted from archived records** ([...] marks elisions). **(a)** Failed trajectory and its diagnosis; the orange box marks the diagnosed B.9 step, where re-reading `result.txt` is taken as verification. **(b)** Failure contract derived from the diagnosis. **(c)** Constructed environment and the full inserted note: the only change to the source task is one inserted file (teal); instruction and verifier are byte-identical. **(d)** Static validation checks of the source–target pair.

Table 5: **Hyperparameters of RFT and curriculum GRPO.** †Framework default, not set explicitly in the training configuration.

Hyperparameter	RFT	Curriculum GRPO
Model and data		
Initialization	Qwen3.6-27B	RFT checkpoint (step 392)
Reference model	—	RFT checkpoint (frozen)
Environments	225 (184 with retained trajectories)	225
Objective	Token NLL on verifier-passing trajectories	Group-normalized binary verifier reward
Optimization		
Optimizer	AdamW	AdamW
AdamW $(\beta_1, \beta_2, \epsilon)$	$(0.9,\ 0.95,\ 10^{-8})^{\dagger}$	$(0.9,\ 0.95,\ 10^{-8})$
Weight decay	0.1	0.1
Gradient clipping	$1.0^{\dagger}$	1.0
Peak learning rate	2×10^{-6}	1×10^{-6}
Learning-rate schedule	Cosine, warm-up ratio 0.1	Cosine, no warm-up
Effective batch	32 sequences	24 rollouts (nominal; 23–24 after invalid-group exclusion)
Optimizer updates	392	525 (75 per round)
GRPO objective		
Group size K	—	8
Advantage	—	Group mean and std, $\varepsilon_A = 10^{-4}$
Policy-ratio clip	—	0.2
KL coefficient	—	0.001
KL estimator	—	$e^{\delta} - \delta - 1$ (k3), in the loss
Execution		
Seed	42	1
Hardware	2×8 H200	4×8 H200

B Training Details

This appendix specifies the training pool, the RFT data and objective, the hyperparameters of both training stages (Section 3.3), the curriculum scheduling rules and the resulting dynamics, and the direct GRPO run on Qwen3.5-4B.

B.1 Training pool and base sampling

The pool contains the 225 environments of Section 4.4. The base policy sampled eight rollouts per environment (1,800 in total) with temperature 1.0, top-p 0.95, top-k 20, at most 32,768 tokens per response and 65,536 policy tokens per trajectory, a 1,200-second agent limit, and a 240-second verifier limit. These rollouts yield 1,213 successes and 587 failures.

B.2 RFT dataset and objective

All 1,213 verifier-passing trajectories are retained. They cover $|\mathcal{E}^+| = 184$ environments; the remaining environments contribute no supervised data. Parsing is applied per turn target, and 14 turn targets that fail to parse are dropped. The loss covers only the model's own sampled tokens; system prompts, user inputs, and tool observations are masked, and no target is truncated. With K_e the number of retained trajectories of environment e and $L_{e,j}$ the number of supervised tokens over all turn targets of trajectory j,

$$\mathcal{L}_{\text{RFT}}(\theta) = -\frac{1}{|\mathcal{E}^+|} \sum_{e \in \mathcal{E}^+} \frac{1}{K_e} \sum_{j=1}^{K_e} \frac{1}{L_{e,j}} \sum_{t=1}^{L_{e,j}} \log \pi_\theta\big(y_{e,j,t} \mid x_{e,j,<t}\big). \tag{3}$$

The resulting dataset contains 12,532 turn targets and 6,278,097 supervised tokens, trained for one epoch (392 optimizer steps).

B.3 HYPERPARAMETERS AND EXECUTION

Table 5 lists the settings of both stages.

B.4 CURRICULUM SCHEDULING RULES

Curriculum GRPO starts from the RFT checkpoint and runs $R = 7$ rounds of $N = 225$ groups with $K = 8$ rollouts each. In Eq. (2), $\hat{A}_i = \big(r_i - \mathrm{mean}(\{r_k\}_{k=1}^K)\big)/\big(\mathrm{std}(\{r_k\}_{k=1}^K) + \varepsilon_A\big)$ over the group of τ_i; $\rho_{i,t} = \pi_\theta(y_{i,t} \mid \cdot)/\pi_{\theta_{\text{old}}}(y_{i,t} \mid \cdot)$, where $\pi_{\theta_{\text{old}}}$ is the policy at the start of the round, since each round's rollouts are sampled once and consumed over 75 updates; and $\hat{\mathbb{D}}_{i,t} = e^{\delta_{i,t}} - \delta_{i,t} - 1$ with $\delta_{i,t} = \log \pi_{\text{RFT}}(y_{i,t} \mid \cdot) - \log \pi_\theta(y_{i,t} \mid \cdot)$. Only the policy's own tokens enter $|\tau_i|$ and the loss. Each update uses $B \leq 24$ trajectories (Table 5).

- **Round 1.** Every environment is sampled exactly once under the RFT policy. Base success rates are not reused as difficulty estimates.
- **Rounds 2–7: revisits.** $\lceil N/4 \rceil = 57$ groups go to distinct environments: first those without a valid difficulty record, then the least recently visited, with ties broken by a seeded shuffle.
- **Rounds 2–7: weighted draws.** The remaining 168 groups are drawn one at a time from all environments with fewer than three groups in the round, including those already chosen as revisits. An environment has weight 4 if its latest valid group succeeded on 1–6 of 8 rollouts and weight 1 otherwise (0/8, 7–8/8, or no record). The weight is a relative sampling weight, not a guaranteed ratio.
- **Cap and seeds.** At most three groups (24 rollouts) per environment per round, revisits included. Round t uses seed $42 + t$.
- **Difficulty update.** Records are updated only after every group of the round has terminated. A group containing an infrastructure-invalid rollout does not update difficulty and is excluded from the policy update. When an environment has several valid groups in a round, the last group by pre-assigned index determines its record; results are not averaged. Unvisited environments keep their previous record.
- **Freezing.** Each round's group manifest is frozen before its first rollout, and all groups of a round are sampled from one synchronized policy version.

B.5 PER-ROUND CURRICULUM STATISTICS

Table 6 reports the seven curriculum rounds, including the outcome of every group. Group outcomes are derived from the frozen round manifests and the recorded difficulty updates. The informative rates in Figure 7 omit the seven groups of rounds 2–7 whose environment had no valid difficulty record. Groups whose rollouts all succeed carry zero advantage and account for 51% of valid groups in rounds 2–7 (Table 6). We nevertheless keep mastered environments in the pool rather than filtering them as in dynamic sampling (Yu et al., 2025): every environment reproduces a failure of the base policy, so a single solved group does not show that the decision has been learned. Down-weighting instead of removing them lets a later regression make an environment informative again.

Uniform-sampling estimate. For round r, we classify every environment in the pool by its difficulty record at the start of the round (0/8, 1–6/8, 7–8/8, or no record). The expected informative share under uniform sampling is the sum over classes of the class's share of the pool times the informative rate of that class among the valid groups actually sampled in round r; for a class not sampled in that round, the rate pooled over rounds 2–7 is used. The estimate assumes that an environment's chance of an informative group depends only on its class, not on whether the scheduler selected it.

B.6 DIRECT GRPO ON QWEN3.5-4B

TRACEGYM-4B applies GRPO directly to the official Qwen3.5-4B weights, without RFT; the same official checkpoint serves as the frozen reference model, so π_{RFT} in the KL estimator is replaced by it. Training uses the 225 environments of Appendix B.1, and every training rollout is sampled online from the 4B policy; no Qwen3.6-27B rollouts or difficulty records are reused. It runs a single round of $N = 225$ groups with $K = 8$ rollouts each (1,800 rollouts), in which every environment is

Table 6: **Curriculum GRPO rounds.** Step: optimizer step at the start of the round. Envs.: distinct environments sampled in the round. Weight 4: groups from environments whose latest valid group succeeded on 1–6 of 8 rollouts. Mixed: valid groups with between 1 and 7 successes; All succeed / All fail: valid groups with 8 or 0 successes. Excluded: groups with an infrastructure-invalid rollout, left out of the update. Success: successes among valid rollouts; it differs slightly from the round means in Figure 8, which count only rollouts entering the policy loss.

		Sampling		Valid groups				
Round	**Step**	Envs.	Weight 4	Mixed	All succeed	All fail	**Excluded**	**Success (%)**
Round 1 every environment sampled once								
1	0	225	0	84	113	23	5	74.4
Rounds 2–7 revisits and success-weighted draws								
2	75	138	105	115	84	16	10	70.9
3	150	140	90	96	107	17	5	76.0
4	225	136	83	94	112	15	4	78.2
5	300	138	83	84	116	21	4	77.3
6	375	137	64	80	124	15	6	80.4
7	450	138	56	74	127	20	4	79.2

sampled once under the official weights; the success-adaptive draws of Appendix B.4, which begin in round 2, are therefore not used.

Optimizer, learning rate, clipping, KL coefficient and estimator, and advantage normalization follow the curriculum GRPO column of Table 5, and the sampling, token, and time limits are unchanged from the Qwen3.6-27B run. All language-model parameters are trained, and the vision encoder and aligner are frozen.

C Evaluation

This appendix gives the evaluation protocol and baseline sources (Appendix C.1) and the failure-mode diagnosis (Appendix C.2).

C.1 Evaluation protocol and baseline details

Protocol. Base, Qwen3.5-4B, and our checkpoints are evaluated on the 89 tasks of Terminal-Bench 2.1 with Harbor 0.20.0 and the standard Terminus-2 agent, using remote task containers and the task-native agent and verifier time limits (no time multiplier). Sampling uses temperature 1.0, top-p 0.95, top-k 20, at most 81,920 generated tokens per response, a 262,144-token context, and thinking mode enabled. Each trial uses one fixed seed: Base and TraceGym-27B are each evaluated with five trials. No trial is retried, and attempts that raise any error, including agent timeouts, score zero and remain in the denominator. The Final checkpoint (round 7, optimizer step 525) was fixed before any benchmark result was observed. Confidence intervals resample the 89 tasks with replacement and keep each task's trials paired across the two compared models.

Per-trial results. Across its five trials, Base ranges from 46.1% to 55.1%; TraceGym-27B ranges from 55.1% to 57.3%.

Comparability and exposure. The failure-collection run of Section 4.1 used local containers, a twelve-fold agent time multiplier, and different sampling settings, and it excluded rather than scored timed-out trials. Its success rate is therefore not comparable to Table 2. No Terminal-Bench task is used for training, but failure evidence was collected on Terminal-Bench tasks, so the benchmark is not free of development exposure for TraceGym.

TerminalWorld-Verified. TerminalWorld (Chu et al., 2026) reconstructs terminal tasks from real asciinema recordings; its Verified split contains 200 tasks that were manually reviewed end to end, and we evaluate on all of them. Each model makes one attempt per task with the agent, sampling settings, and time limits of the Terminal-Bench 2.1 protocol above; errors count as failures. Confidence

intervals resample the 200 tasks with replacement, pairing each task across the two compared models. The scores published with TerminalWorld use the same split but a single attempt at temperature 0.7 and no turn limit, so we do not compare against them.

Baseline results. We did not re-evaluate baselines; all baseline scores in Table 2 are taken from the literature. Scores of proprietary models and GLM-5.1 are taken from the public Terminal-Bench 2.1 leaderboard (Terminal-Bench, 2026) (89 tasks $\times$ five trials; Terminus-2 entries where available). Scores of DeepSeek-V4-Pro-Preview, Kimi-K2.6, Qwen3.5-27B, Qwen3.5-9B, and FACET-27B were measured by FACET under Terminus-2 and averaged over three attempts per task (Table 2 of Shi et al. (2026)); FACET distills DeepSeek-V4-Pro (DeepSeek-AI, 2026) trajectories into Qwen3.5-27B. RST applies PPO to Qwen3.5-27B on synthesized tasks, initialized from the base weights, and reports the mean of three runs with Terminus-2 hosted by Harbor, without thinking mode, on the 89-task Terminal-Bench 2, which we take to be Terminal-Bench 2.1 (Li et al., 2026). Tmax applies DPPO without prior supervised fine-tuning, selects checkpoints on Terminal-Bench Lite, and reports the mean of three runs with a harness based on mini-SWE-agent (Ivison et al., 2026). Because these settings differ from ours and from each other (the same Qwen3.6-27B scores 40.5%, 53.9%, and 48.8% in the Tmax and FACET papers and under our protocol), we compare methods by their gain over their own initialization within each source. We include training methods that report Terminal-Bench 2.1 scores together with their initialization; methods that report only earlier benchmark versions are not listed.

C.2 Failure-mode diagnosis

This subsection details the blinded failure-mode diagnosis behind Section 5.3.

Cohort. To limit diagnosis cost, Section 5.3 uses only trials 1–2, comprising 89 paired tasks and 178 attempts per model; no later trial enters this analysis. Base has 83 successes, 41 ordinary failures and 54 agent timeouts; TraceGym-27B has 100 successes, 25 ordinary failures and 53 timeouts. Native outcomes are preserved, including errors scored zero, and no benchmark attempt is rerun.

Diagnosis procedure. We use the fixed 30-code taxonomy and two-stage ATLAS procedure (failure-point analysis, then code mapping) with GPT-5.6-Sol, temperature 1.0, four workers and a 300-second per-call timeout. The target ten-code set is fixed before diagnosis. Inputs withhold arm, model identity, reward, evaluation-verifier output and the target-code set, while retaining feedback actually visible to the agent. Each trajectory uses a uniform projection of at most 48 steps and 128 evidence references, with up to 32 slots reserved for original model-response summaries. Original responses and native parser feedback distinguish valid protocol output from its ATIF rendering; omitted steps do not establish that an action never occurred. Full prompts, responses and evidence references are archived.

Rates and missing-label bounds. For each code with h observed hits and u unsuccessful attempts without a valid diagnosis, the per-attempt rate is h/N with $N = 178$, and the failure-conditional rate lies in $[h/F, (h+u)/F]$, where F is the number of unsuccessful attempts (95 for Base; 78 for TraceGym-27B). Successful attempts are not diagnosed, and one failure can carry several codes, so these are occurrence counts, not exclusive causes of failure.

Statistical inference. Confidence intervals use 10,000 paired task-cluster bootstrap draws that keep both trials and both models of a task together; they reflect task resampling only and are not adjusted for multiple comparisons. Table 7 reports the summary contrasts and Table 8 all thirty code changes. The interval for the targeted-minus-non-targeted change, $[-5.73, +1.40]$, includes zero. Relative to Base, the targeted and non-targeted means fall by 17.0% and 22.8%, close to the 17.9% reduction in unsuccessful attempts, so the gap between them is also compatible with a reduction in failures that is uniform across codes.

D Compute and token usage

Table 9 reports the token usage and GPU time of each stage. Constructing one statically valid environment (source-task selection and artifact generation) takes on average 130K input tokens, of which

Table 7: **Measured two-trial Q2 summary.** Both arms contain 178 attempts. All differences are TRACEGYM-27B minus Base; the last row subtracts the non-targeted mean change from the targeted mean change. Code rates count observed labels.

	Rate (%)		Change (pp)	
Metric	Base	TRACEGYM-27B	Δ	95% CI
Unsuccessful outcomes				
Agent timeout	30.34	29.78	−0.56	[−4.49, +3.37]
Ordinary failure	23.03	14.04	−8.99	[−15.17, −3.37]
Failure-code rates				
Targeted (mean)	24.72	20.51	−4.21	[−8.20, −0.39]
Non-targeted (mean)	9.38	7.25	−2.13	[−3.51, −0.76]
Targeted minus non-targeted	—	—	−2.08	[−5.73, +1.40]

Table 8: **All thirty failure codes in the two-trial Q2 cohort,** grouped by taxonomy category; bold codes are targeted. *Hits*: observed unsuccessful attempts with the code as a primary or secondary label. *Change*: per-attempt rate change with its 95% interval, using the denominator 178. *Failure-conditional*: rate among unsuccessful attempts (denominators 95 and 78), shown as missing-label bounds where diagnoses are missing.

	Hits n		Change (pp)		Failure-conditional (%)	
Code	Base	TRACEGYM-27B	Δ	95% CI	Base	TRACEGYM-27B
A System failures						
A.1	36	29	−3.93	[−8.99, +0.56]	37.89	[37.18, 39.74]
A.2	9	3	−3.37	[−6.74, −0.56]	9.47	[3.85, 6.41]
A.3	18	7	−6.18	[−11.80, −1.12]	18.95	[8.97, 11.54]
A.4	13	10	−1.69	[−5.62, +2.25]	13.68	[12.82, 15.38]
A.5	19	14	−2.81	[−7.87, +2.81]	20.00	[17.95, 20.51]
A.6	36	24	−6.74	[−12.92, −0.56]	37.89	[30.77, 33.33]
A.7	23	13	−5.62	[−10.67, −0.56]	24.21	[16.67, 19.23]
A.8	11	6	−2.81	[−7.30, +1.69]	11.58	[7.69, 10.26]
A.9	13	13	+0.00	[−4.49, +4.49]	13.68	[16.67, 19.23]
A.10	75	64	−6.18	[−14.04, +1.12]	78.95	[82.05, 84.62]
A.11	63	49	−7.87	[−14.04, −1.69]	66.32	[62.82, 65.38]
B Solver quality failures						
B.1	33	29	−2.25	[−8.43, +3.93]	34.74	[37.18, 39.74]
B.2	22	19	−1.69	[−7.30, +3.93]	23.16	[24.36, 26.92]
B.3	30	26	−2.25	[−9.55, +5.06]	31.58	[33.33, 35.90]
B.4	17	21	+2.25	[−3.93, +8.43]	17.89	[26.92, 29.49]
B.5	30	26	−2.25	[−7.87, +3.37]	31.58	[33.33, 35.90]
B.6	29	24	−2.81	[−8.99, +3.37]	30.53	[30.77, 33.33]
B.7	13	18	+2.81	[−2.25, +7.87]	13.68	[23.08, 25.64]
B.8	67	51	−8.99	[−17.98, −0.56]	70.53	[65.38, 67.95]
B.9	31	22	−5.06	[−11.80, +1.69]	32.63	[28.21, 30.77]
B.10	66	54	−6.74	[−14.61, +1.12]	69.47	[69.23, 71.79]
B.11	56	51	−2.81	[−9.55, +3.93]	58.95	[65.38, 67.95]
C Domain reasoning failures						
C.1	40	26	−7.87	[−14.61, −1.12]	42.11	[33.33, 35.90]
C.2	9	8	−0.56	[−5.06, +3.37]	9.47	[10.26, 12.82]
C.3	2	6	+2.25	[−0.56, +5.62]	2.11	[7.69, 10.26]
C.4	2	2	+0.00	[+0.00, +0.00]	2.11	[2.56, 5.13]
C.5	3	3	+0.00	[+0.00, +0.00]	3.16	[3.85, 6.41]
C.6	2	2	+0.00	[+0.00, +0.00]	2.11	[2.56, 5.13]
C.7	2	2	+0.00	[−2.25, +2.25]	2.11	[2.56, 5.13]
C.8	4	1	−1.69	[−3.93, +0.00]	4.21	[1.28, 3.85]

105K are cache hits, and 4.4K output tokens of GPT-5.6-Sol; no GPU is needed until the learner is rolled out. Training TRACEGYM-27B, including base sampling, RFT, and curriculum GRPO, takes 1,353 H200-hours. Input tokens are summed over requests and therefore count repeated context across the turns of a trajectory. H200-hours are allocated GPU time (GPUs $\times$ wall time), including model loading and idle time. We report no monetary cost, because provider prices were not recorded.

Table 9: **Compute and token usage.** TB2.1: Terminal-Bench 2.1. Tokens in millions; Cached: cached input tokens, where recorded. H200-h: allocated H200 GPU-hours; — for hosted API calls or where the allocation was shared with other runs. For RFT, the input column gives supervised tokens. Artifact generation counts the 2,439 of 2,500 calls with usage records, and learner rollouts the 2,055 of 2,110 rollouts with token counters. ‡Estimated by scaling the token counts of trials 1–2 to all five trials.

			Tokens (M)			GPU
Stage	**Model**	**Volume**	Input	Cached	Output	H200-h
Construction						
Failure diagnosis	GPT-5.6-Sol	289 calls	9.7	—	0.9	—
Source-task selection	GPT-5.6-Sol	185 calls	114.1	98.6	2.2	—
Artifact generation	GPT-5.6-Sol	2,500 calls	159.3	123.9	7.1	—
Learner rollouts	Qwen3.6-27B	2,110 rollouts	246.2	—	19.0	1,163
Re-diagnosis of failures	GPT-5.6-Sol	831 calls	15.9	—	2.2	—
Training						
Base sampling	Qwen3.6-27B	1,800 rollouts	190.3	—	7.8	76
RFT	Qwen3.6-27B	392 steps	6.3	—	—	102
Curriculum GRPO	Qwen3.6-27B	12,600 rollouts	2,125.0	—	81.7	1,175
Evaluation						
TB2.1 evaluation	Qwen3.6-27B	445 attempts	292.2	—	16.8	470
TB2.1 evaluation	TRACEGYM-27B	445 attempts	335.1‡	—	15.5‡	—
Q2 diagnosis	GPT-5.6-Sol	156 calls	5.3	—	0.5	—