From Patches to Trajectories: Privileged Process Supervision for Software-Engineering Agents

Murong Ma[1]* **Tianyu Chen**[2]† **Yun Lin**[3]† **Shuai Lu**[2] **Qinglin Zhu**[4]
Yeyun Gong[2]† **Zhiyong Huang**[1] **Peng Cheng**[2] **Yan Lu**[2] **Jin Song Dong**[1]
[1]National University of Singapore [2]Microsoft Research Asia
[3]Shanghai Jiao Tong University [4]King's College London

Abstract

Supervised fine-tuning (SFT) on long teacher trajectories is the dominant method for instilling investigation and reasoning capabilities into open software-engineering (SWE) agents. Under SFT, every retained response is an imitation target, so the student inherits not only the trajectory's outcome but also any flaw in its intermediate steps, including ungrounded leaps and redundant loops. High-quality training data must therefore be jointly *effective* (each step is grounded and narrows the agent's epistemic gap to the correct fix) and *efficient* (each step is information-bearing rather than redundant or looping). Existing recipes filter or relabel teacher rollouts using only a binary terminal verifier, which does not directly target these axes and provides no supervision on instances where the teacher fails.

Every real issue ships with a developer-authored reference patch $p^\star$ that implicitly testifies to the file paths, runtime behaviors, and conventions a fix presupposes, but the standard pipeline discards it. We propose **P2T** (Patches-to-Trajectories), which uses $p^\star$ as *privileged information* during curation, and frames trajectory construction as a bi-objective program over per-step effectiveness and trajectory length. A *reverse* phase distills $p^\star$ into a latent *process graph* $G^\star$ of contextual facts and solution milestones, encoding dense intermediate anchors in constructive ordering. A *forward* phase curates trajectories from blinded teacher continuations, scoring per-step progress against $G^\star$ under a leakage-blocking groundedness check and committing the shortest segments that retain effectiveness.

Using only 1.7k curated SWE-Gym trajectories, P2T improves both axes simultaneously over outcome-filtered SFT and its tool-error-masking variant: on SWE-bench Verified, it lifts Pass@1 by up to $+10.8$ points while cutting per-instance inference cost by $\sim 15\%$, with consistent gains on SWE-bench Lite and across two teachers. A size-matched ablation and qualitative analysis further isolate per-trajectory quality from data scale.

1 Introduction

Autonomous software-engineering agents built on large language models (LLMs) are now routinely competitive on real GitHub issue-resolution benchmarks [5, 2, 30, 27], navigating repositories, localizing faults, editing code, and validating fixes [24, 18]. A capable agent must do more than emit a final patch that happens to pass: it must learn to *investigate*, *reason*, and *validate*, the per-step competencies that make terminal success reproducible rather than incidental. The dominant route to instilling these competencies in open base models is supervised fine-tuning (SFT) on long trajectories from strong teacher models [10, 11, 4, 29], which provides dense process supervision

*Work done during internship at Microsoft Research.

†Corresponding authors: chentianyu@microsoft.com, lin_yun@sjtu.edu.cn, yegong@microsoft.com.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

across turns of ReAct-style interactions [26]. Under SFT—a behavior-cloning objective for sequential decision problems [14]—every retained response is an imitation target, so the student inherits not only the trajectory's outcome but also any flaw in its intermediate steps. Each trajectory therefore needs two complementary properties. **Effectiveness:** each step narrows the agent's epistemic gap to the correct fix by uncovering a fact the fix presupposes, grounded in the visible prefix with no unsupported leaps or premature conclusions. **Efficiency:** each step is information-bearing, advancing the trajectory rather than re-deriving established facts, looping on uninformative actions, or padding with redundant exploration. The two are in direct tension: cautious exploration lengthens trajectories, while aggressive shortening invites unsupported shortcuts. Constructing trajectories on the right side of this tradeoff is the central data problem for SFT of SWE agents.

Existing recipes do not directly target these axes. The standard pipeline samples teacher rollouts and retains only those whose final patch passes the issue's tests [11]; variants scale the instance pool by procedurally synthesizing executable issues [25, 4, 29]. All inherit the same binary terminal verifier: an outcome-supervision signal that supplies feedback only on the final result rather than on intermediate reasoning steps [7]. It is therefore structurally indifferent to either axis. On the SWE-Gym training pool, retained trajectories often exhaust the 100-iteration budget without reaching a normal finish, accidentally tripping the test suite (7.6% under the Qwen3-Coder-480B teacher, 9.3% under GLM-5-FP8); 6.8%/8.6% of their file-viewing actions revisit content already viewed earlier in the trajectory; and 70.2%/64.7% of instances contribute *no* supervision because the teacher never produces a passing patch. Independent audits further show that a non-trivial share of "passing" patches reflect weak tests or solution leakage rather than correctness [1]; more broadly, test-suite-based program repair has long been known to admit plausible but incorrect or overfitted patches [13, 15]. The verifier is therefore not even tight on its own axis. The remaining question is therefore not how to acquire *more* tasks, but how to extract *better* per-trajectory supervision from the real ones.

The signal that addresses both axes is already available, but unused: the developer-authored *reference patch* $p^\star$ associated with each issue–pull-request instance in real-issue SWE benchmarks [5], which enters the standard pipeline only as a discarded ground truth. As process supervision, $p^\star$ is uniquely well-positioned, since each line of it implicitly testifies to the file paths, runtime behaviors, and conventions a solver would have had to uncover before the edit becomes derivable. We therefore propose to use $p^\star$ as *privileged information* [17] during curation: a quantity the data-construction procedure may consult to score and shape trajectories, but that the student never sees. With $p^\star$ in scope, the curator can score per-step progress against the prerequisites a fix presupposes (effectiveness), keep only information-bearing steps (efficiency), and recover supervision precisely on the hard instances where ordinary teacher rollouts fail.

The challenge is that conditioning trajectory generation directly on $p^\star$ leaks the answer: any prefix built with $p^\star$ in scope risks splicing in edits, claims, or file references no honest investigation could yet support, and a student that imitates such a trace internalizes the same unjustified leaps. We therefore propose **P2T** (*Patches-to-Trajectories*), which frames trajectory curation as a *bi-objective program* over per-step effectiveness and trajectory length, and mediates $p^\star$ through a latent *process graph* $G^\star$ distilled from it, so the curator can shape trajectories along both axes without ever exposing $p^\star$ to the student. Empirically, P2T improves both axes simultaneously over outcome-filtered SFT and its tool-error-masking variant: on SWE-bench Verified, it lifts Pass@1 by up to $+10.8$ points while cutting per-instance inference cost by $\sim 15\%$, and a size-matched control already beats both baselines on both axes, isolating per-trajectory quality from data scale. Our contributions are as follows:

- We frame SWE-agent SFT data construction as a *bi-objective* program over per-step effectiveness and trajectory length, and show that outcome-filtered rejection sampling provides no per-step or length signal.
- We propose P2T, a curation framework that uses $p^\star$ as *privileged information*: a reverse phase distills $p^\star$ into a process graph $G^\star$ of contextual facts and solution milestones, and a forward phase realizes trajectories that are short, grounded, and steered by $G^\star$.
- Using only 1.7k curated SWE-Gym trajectories, P2T improves Pass@1 by up to $+10.8$ points while cutting per-instance inference cost by $\sim 15\%$ over outcome-filtered SFT, on SWE-bench Verified and Lite across two students and two teachers.

2 Related Work

SWE agents and benchmarks. SWE-bench [5] catalyzed a line of inference-time systems for repository-level issue resolution: ReAct-style tool use [26], SWE-agent's agent–computer interface [24], the OpenHands platform [19], structure-aware retrieval in AutoCodeRover [31], and the simpler localize–repair–validate pipeline of Agentless [21]. Audits show that terminal pass/fail can overstate correctness when tests are weak or issues leak the solution [1, 20]. Our work is orthogonal to these scaffolds: we improve the per-step *quality* of the SFT trajectories on which such agents are trained.

Trajectory data for open SWE agents. Existing recipes scale executable tasks and retain trajectories that pass a terminal verifier: SWE-Gym [11] on real Python issues, R2E-Gym [4] with procedural construction and hybrid verifiers, SWE-smith [25] via test-breaking synthesis, and Skywork-SWE [29] on large-scale curation and trajectory scaling. All retain whole successful rollouts, inheriting their detours, redundant observations, and unsupported inferences. P2T instead treats $p^\star$ as privileged curation information, distilled into $G^\star$ to expose only the prerequisites a fix presupposes, while never showing $p^\star$ to the student. This follows the broader principle of curating training data against a small trusted reference rather than the noisy signal the data itself supplies [8].

Trajectory and context reduction for LLM agents. A complementary line attacks the inference-time cost of long agent histories: AgentDiet prunes useless, redundant, or expired entries from coding-agent trajectories at run time [22], while ACON learns to compress observations and interaction histories for long-horizon agents [6]. These methods leave the underlying policy fixed and shorten what it consumes; P2T instead shortens what it *produces* at training time, so the resulting student is intrinsically efficient and remains compatible with such inference-time compressors.

3 Problem Setting

We study the construction of process-supervision data for supervised fine-tuning (SFT) of autonomous software-engineering agents. Under SFT, every response in a training trajectory becomes an imitation target, so a trajectory's training value is bounded by its weakest step: a passing terminal patch does not redeem a prefix that contains hallucinated reasoning, redundant exploration, non-progressing action loops, or uninformative tool calls. Data construction must therefore control two complementary properties of each trajectory: *process effectiveness*—every step makes prefix-grounded progress toward the reference fix, and *process efficiency*—the trajectory is short, to limit student inference cost and reduce the surface area of imitation noise. The two are in direct tension: cautious, well-grounded exploration lengthens trajectories, while aggressive shortening invites unsupported leaps. We frame trajectory curation as a *bi-objective optimization* problem over these two criteria.

3.1 Trajectories and the SFT Objective

A task instance is a tuple $\mathcal{I}_i = (d_i,\, R_i,\, E_i,\, \mathcal{T}_i,\, p_i^\star)$, where d_i is the issue description, R_i the repository at the pre-fix commit, E_i a sandboxed execution environment exposing a fixed tool set (file viewing, shell execution, code editing), $\mathcal{T}_i$ the issue's test suite, and $p_i^\star$ the reference fix patch. We split the components of $\mathcal{I}_i$ into the *non-oracle bundle* $x_i := (d_i, R_i, E_i)$ that any solver may consume, and the *oracle bundle* $(p_i^\star, \mathcal{T}_i)$, which the data-construction procedure may use to evaluate trajectories.

The agent interacts with R_i through E_i. At turn t it observes the visible prefix $h_t = (d_i, y_1, o_1, \ldots, y_{t-1}, o_{t-1})$ and emits a ReAct-style response $y_t = (c_t, a_t)$, comprising reasoning c_t and an action a_t; executing a_t in E_i returns an observation o_t. A trajectory is the resulting sequence $\tau_i = (d_i,\, y_1, o_1,\, \ldots,\, y_T, o_T)$, and a constructed collection induces the SFT dataset $\mathcal{D} = \{(h_t, y_t) : (h_t, y_t) \in \tau_i\}$. The student π_θ is trained by behavioral cloning,

$$\mathcal{L}_{\mathrm{SFT}}(\theta) \;=\; -\,\mathbb{E}_{(h_t,y_t)\sim\mathcal{D}}\big[\log \pi_\theta(y_t \mid h_t)\big].$$

Because every retained response becomes a training target, data construction must control not only whether the final patch is correct, but whether the intermediate process is itself worth imitating.

3.2 Outcome-Filtered Trajectory Collection

The dominant paradigm uses $\mathcal{T}_i$ purely as a terminal verifier and discards $p_i^\star$. Given a teacher policy π_T, one samples K trajectories $\tau_i^{(k)} \sim \pi_T(\cdot \mid x_i)$ per instance and retains those whose induced patch $\hat{p}_i^{(k)} = \mathrm{patch}(\tau_i^{(k)})$ passes the test suite:

$$\mathcal{D}_{\mathrm{OF}} \;=\; \big\{\, (h_t, y_t) \in \tau_i^{(k)} \;:\; \mathcal{T}_i\big(\hat{p}_i^{(k)}\big) = 1 \,\big\}.$$

This procedure treats terminal success as the only supervision signal. As a result, retained data might include low-quality trajectories, i.e. failing to distinguish a concise, evidence-driven solution from one that succeeds after redundant search, unsupported claims, or accidental edits.

3.3 Patch-Oracled Bi-objective Trajectory Construction

To avoid such low-quality trajectory, in this paper, we propose to utilize the reference patch $p_i^\star$, which encodes precisely how a competent developer resolves the issue as a source of process supervision. Specifically, we make $p_i^\star$ a process oracle: a trajectory is judged by whether its steps uncover the evidence—file and symbol localizations, runtime behavior, and implementation choices—needed to derive a fix equivalent to $p_i^\star$. This admits any trajectory that establishes the right intermediate evidence, and rejects trajectories whose patch happens to pass.

Bi-objective trajectory target. We score each trajectory along two axes. *Process effectiveness* $\mathrm{Eff}_i(\tau) \in [0, 1]$ rewards steps that uncover fix-relevant evidence without leaping ahead of what the prefix supports; we keep it abstract here and instantiate it in Sec. 4 via a *process graph* $G_i^\star$ distilled from $p_i^\star$. *Process efficiency* is the trajectory length in generated response tokens, $\mathrm{Len}_i(\tau)$. When given a set of trajectories $\mathcal{T}(\mathcal{I}_i)$ on task $\mathcal{I}_i$, we pick our target one by the *shortest-above-floor* rule: among trajectories whose effectiveness clears a calibrated floor η_i, take the shortest (a standard ε-constraint scalarization of bi-objective programs [9]); the chosen trajectory is admitted into the SFT dataset only if its final patch passes the test suite $\mathcal{T}_i$:

$$\tau_i^\star \;=\; \arg\min_{\tau \in \mathcal{T}(\mathcal{I}_i)} \mathrm{Len}_i(\tau) \quad \text{s.t.} \quad \mathrm{Eff}_i(\tau) \geq \eta_i, \qquad \mathcal{D}_{\mathrm{ours}} \;=\; \{(h_t, y_t) \in \tau_i^\star\}_{i=1}^N.$$

Open challenges. This formulation leaves two questions for Sec. 4: (i) how to anchor *process effectiveness* so that it captures fix-relevant progress, is sensitive to ungrounded leaps, and is not itself a leakage channel for $p_i^\star$; and (ii) how to operationalize the resulting bi-objective program tractably, given that $\mathcal{T}(\mathcal{I}_i)$ cannot be searched exhaustively.

4 Method

We instantiate the bi-objective program of Section 3.3 with a two-phase pipeline that resolves its two open challenges in turn:

- **Phase 1: Process Graph Distillation (Sec. 4.1).** We distill $p_i^\star$ into a latent *process graph* $G_i^\star$ whose nodes name the intermediate contextual facts and solution milestones that must be established before the fix becomes derivable, and on top of $G_i^\star$ we define a *progress* score that measures how much of the graph a trajectory legitimately uncovers—this resolves challenge (i).
- **Phase 2: Receding-Horizon Bi-Objective Trajectory Realization (Sec. 4.2).** We grow the trajectory one segment at a time via a sliding window. Within each window, we sample a set of candidate segments and apply the same shortest-above-floor rule from Sec. 3.3 locally, then repeat on the extended prefix—this resolves challenge (ii).

Figure 1 illustrates the pipeline.

4.1 Phase 1: Process Graph Distillation

The reference patch $p_i^\star$ defines the *target state* of the repository, but does not itself describe a valid *discovery process*: conditioning trajectory generation directly on $p_i^\star$ would let edits and unsupported

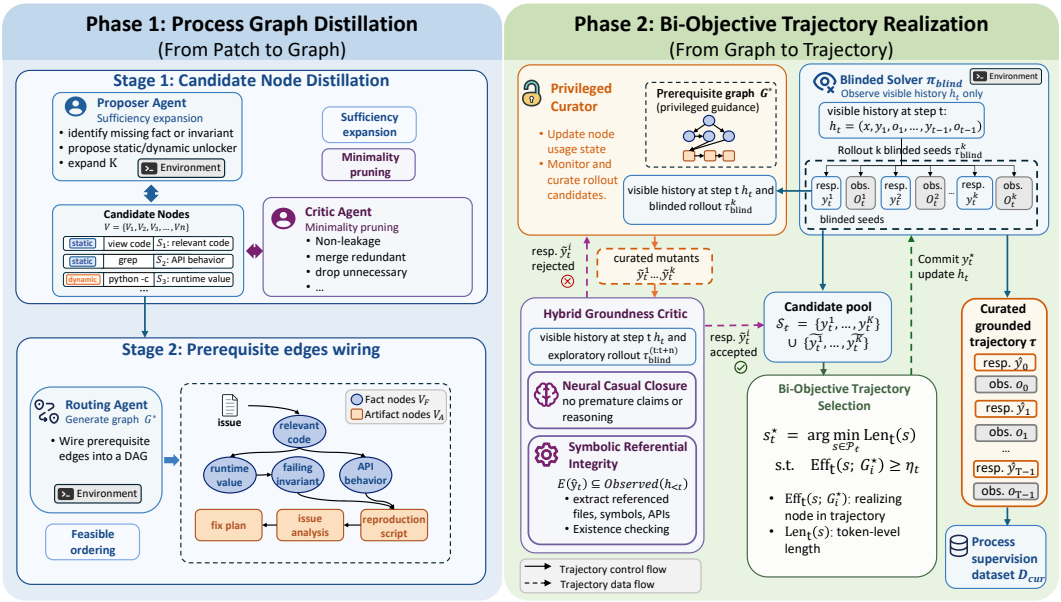


Figure 1: Overview of P2T. Phase 1 (Sec. 4.1) distills the reference patch $p_i^\star$ into a minimal latent process graph $G_i^\star$ of contextual facts and solution milestones—including the eventual edits and validations—that any solver would need to uncover. Phase 2 (Sec. 4.2) realizes a trajectory by sampling blinded continuations, locally mutating them toward currently available graph nodes, and committing the segment that achieves both effectiveness and efficiency (defined in Sec. 3.3).

leaps appear in the prefix before the agent has any evidence to justify them. We therefore convert $p_i^\star$ into a latent process graph

$$G_i^\star \;=\; (V_i,\, E_i),$$

in which each node $v \in V_i$ names an intermediate contextual fact or solution milestone that any solver would need to establish before the fix becomes derivable, and each edge encodes a prerequisite relation. $G_i^\star$ is the structure on which the per-step effectiveness signal is defined.

Node format. Each node is represented as $v = (s_v, \eta_v, u_v)$, where s_v is a natural-language statement, η_v a type tag, and u_v an explicit *unlocker*: the environment interaction needed to discover s_v. We use two kinds of nodes. *Contextual-fact nodes* record claims about the repository or its runtime behavior whose discovery is a prerequisite for fixing the issue; their unlockers are either *static* (no execution required, e.g., reading a file, inspecting a class hierarchy, or a repository-wide `grep`) or *dynamic* (require execution, e.g., running a test, evaluating a probe script, or inspecting runtime values). *Solution-milestone nodes* record the intermediate products an agent must construct on the way to the fix—reproduction scripts, root-cause analyses, fix plans, code edits, and validation runs—whose unlockers are the corresponding tool calls (writing a script, drafting an analysis or plan, applying an edit, running the test suite).

Three desiderata for $G_i^\star$. We require $G_i^\star$ to be jointly: (i) *sufficient*—the issue, repository, and graph nodes together make $p_i^\star$ plausibly derivable, so that no essential localization, behavioral, or implementation fact is omitted; (ii) *non-leaking*—each node's unlocker is conceivable from the issue, repository, and the node's predecessors alone, so that proposing the unlocker does not presuppose knowledge of $p_i^\star$ (e.g. an edit node may not appear before its motivating root-cause-analysis and fix-plan nodes); (iii) *feasibly ordered*—the graph admits a topological order realizable through ordinary environment interaction, with each node discoverable only after its prerequisites are established.

Instantiation. We construct $G_i^\star$ by an iterative *proposer–critic* procedure implemented with two specialized LLM agents. Starting from $V_i^{(0)} = \varnothing$, the proposer adds candidate nodes that close the remaining logical gap to $p_i^\star$, each annotated with a candidate unlocker; this targets desideratum (i), *sufficiency*. The critic then prunes any candidate whose declared unlocker is not motivated by the nodes already in $V_i^{(t)}$—for instance, an edit node introduced before any root-cause-analysis or fix-

plan node—and emits feedback indicating which aspects remain under-determined; this enforces desideratum (ii), *non-leakage*. The loop terminates when the node set stabilizes. A final organization step then links the surviving nodes into the DAG $G_i^\star$ by drawing a prerequisite edge from u to v whenever u must be established before v's unlocker can apply, enforcing desideratum (iii), *feasible ordering*. Full prompts are deferred to App. D; a worked example on a real SWE-Gym instance is given in App. I.

Node establishment. We say a node v is *established* by a trajectory prefix h_t, written $\mathrm{Est}(v, h_t) = 1$, when both: (a) some action in h_t matches requirement specified by u_v (e.g., a repository-wide `grep` for an unlocker requiring a `grep` action); and (b) an LLM verifier, conditioned only on the text of h_t, judges that the resulting observations entail the statement s_v. Restricting the verifier to h_t—rather than letting it probe the repository on its own—ensures that establishment reflects what the trajectory has actually surfaced, not what is in principle knowable. We write $U_t = \{v \in V_i : \mathrm{Est}(v, h_t) = 1\}$ for the established set at step t. The verifier prompt is deferred to App. H.

Graph-aware progress. We want each step of a trajectory to advance *coverage* of $G_i^\star$—establishing more of its nodes, in dependency-respecting order—without leaping to a solution-milestone move (fix plan, edit, validation) before the contextual facts it presupposes have been established. We capture both desiderata in a single per-step score $\mathrm{Prog}_t \in [0, 1]$ that rewards new coverage and zeroes out the moment a node is established prematurely. Formally, at step t, let

$$\mathcal{A}_{t-1}(G_i^\star) \;=\; \big\{\, v \in V_i \setminus U_{t-1} \;:\; \mathrm{Pred}_{G_i^\star}(v) \subseteq U_{t-1} \,\big\}$$

be the *available frontier*: nodes legitimately discoverable from h_{t-1}. Then we define

$$\mathrm{Prog}_t \;=\; \frac{|U_t \setminus U_{t-1}|}{\max(|\mathcal{A}_{t-1}(G_i^\star)|,\, 1)} \;\cdot\; \mathbb{1}\big[\, U_t \setminus U_{t-1} \subseteq \mathcal{A}_{t-1}(G_i^\star) \,\big] \;\in\; [0, 1].$$

The numerator counts newly established nodes; the denominator normalizes by what was eligible to be established; and the indicator hard-zeros the score if any newly established node is non-discoverable from h_{t-1}. Prog_t is the per-step backbone on which Phase 2 builds segment-level effectiveness Eff_t, by aggregating across a segment's steps and composing with a complementary groundedness gate (Sec. 4.2).

4.2 Phase 2: Receding-Horizon Bi-Objective Trajectory Realization

Phase 1 supplies $G_i^\star$ and the per-step backbone Prog_t; Phase 2 turns them into an executable trajectory. We grow the trajectory one segment at a time (Fig. 1) and apply the bi-objective rule of Sec. 3.3 segment by segment: at each step t we form a small pool of length-n candidate segments, commit the shortest whose effectiveness clears a local floor η_t, and replan from the updated prefix. Both Eff and Len are additive over the segment partition, so under a matching floor calibration ($\sum_t \eta_t = \eta_i$) these greedy commits realize the trajectory-level shortest-above-floor rule on the family of rollouts the per-window pools induce (See proof sketch in App. E).

Candidate generation: blind seed plus single mutation. At each window we draw K length-n seeds from the blinded solver, $\tilde{s}_t^{(k,0)} \sim \pi_{\mathrm{blind}}^{(n)}(\cdot \mid h_t)$, so every seed is on-prefix and on-distribution. A pure seed may miss the next available frontier node of $G_i^\star$; the curator is therefore allowed to perturb at most one of its steps—picking a position $t + j$ and a target $v \in \mathcal{A}_{t+j}(G_i^\star)$, proposing a replacement response y'_{t+j} under prefix h_{t+j}, and letting π_{blind} re-roll the suffix. The window-t candidate pool $\mathcal{S}_t$ is the union of the K pure seeds and their single-edit graph-aware variants (App. F). Because every candidate is a blinded continuation modulo at most one localized rewrite, the pool stays close to the student distribution while remaining steerable by $G_i^\star$, and any $(\mathrm{Eff}_t, \mathrm{Len}_t)$ gap between a seed and one of its variants is attributable to that single edit.

Per-segment effectiveness. We assemble the per-segment effectiveness Eff_t demanded by Sec. 3.3 by aggregating the per-step progress over the segment, and composing with a binary groundedness gate,

$$\mathrm{Eff}_t(s;\, G_i^\star) \;=\; \Big(\sum_{\tau \in s} \mathrm{Prog}_\tau\Big) \;\cdot\; \mathrm{Ground}_t(s), \qquad \mathrm{Ground}_t \in \{0, 1\}.$$

The gate applies only to the curator-introduced edit: blinded seeds are by construction on-prefix, so we set $\mathrm{Ground}_t(s) = 1$. A failure sends $\mathrm{Eff}_t(s)$ to zero, removing the candidate from contention regardless of how much progress it appears to make. The binary leakage rejection inside Prog_t zeros

out any step in the segment with a premature establishment, while the gate inspects only the mutated step itself, so the two operate on disjoint scopes within a candidate.

Groundedness. For a mutated step y under prefix h, let $\mathrm{Ents}(y)$ be the repository entities (file paths, identifiers, function and class names) it references and let $\mathrm{Obs}(h)$ be the entities that have appeared in observations within h; a symbolic referential-integrity check passes when $\mathrm{Ents}(y) \subseteq \mathrm{Obs}(h)$, blocking references to an as-yet-unseen entity. A complementary LLM judge returns $\mathrm{Claim}(y, h) \in \{0, 1\}$ on whether the reasoning of y is entailed by the observations in h, ruling out premature root-cause assertions and unsupported leaps. For a mutated candidate with edit at position $t + j$,

$$\mathrm{Ground}_t(s) \;=\; \mathbb{1}\big[\mathrm{Ents}(y'_{t+j}) \subseteq \mathrm{Obs}(h_{t+j})\big] \;\cdot\; \mathrm{Claim}(y'_{t+j},\, h_{t+j}),$$

i.e., the mutated step must clear both the symbolic check (cheap, catches concrete entity leakage) and the neural judge (catches semantic leaps that are syntactically grounded but logically unsupported). Entity-extractor patterns and the LLM-judge prompt are deferred to App. H.

Selection and commit. Segment length is measured either as response-token mass $\mathrm{Len}_t(s) = \sum_{y \in s} |y|$ or as step count $\mathrm{Len}_t(s) = |s|$. Among all candidate segments whose effectiveness clears a local threshold η_t, we commit the shortest (following the shortest-above-floor rule in Sec. 3.3),

$$s_t^\star \;=\; \arg\min_{s \in \mathcal{S}_t} \mathrm{Len}_t(s) \quad \text{s.t.} \quad \mathrm{Eff}_t(s;\, G_i^\star) \geq \eta_t,$$

falling back to the segment of maximum effectiveness if no candidate clears the floor. To avoid locking in a stale suffix, we adopt a half-segment stride: only the first half of $s_t^\star$ is appended to the prefix before the procedure replans, so consecutive critic windows overlap by half their length (e.g., a window over steps 1–10 is followed by one over 6–15). The procedure terminates when the agent submits a patch; trajectories whose patch fails the test suite $\mathcal{T}_i$ are discarded. The end-to-end pipeline is summarized in Algorithm 1 (App. G).

5 Experiments

5.1 Experimental Setup

Training instances. We draw the training pool from SWE-Gym [11] (2,438 real Python issue-resolution instances over 11 repositories), keeping the 1.8k instances with a working Docker environment, a reference patch that passes its tests, $\mathcal{T}_i(p_i^\star) = 1$ (required because P2T consumes $p_i^\star$ as privileged information), and an admissible process graph; P2T curates 1.7k trajectories from this pool that pass final verification (App. N). Rejection-sampling baselines, which never inspect $p_i^\star$, run on the full 2,126 executable instances. The reverse phase yields 33,106 graph nodes (median 18 per instance), dominated by static and dynamic facts; full breakdown in App. B.

Curators, scaffold, students, baselines. We curate trajectories with two teachers, Qwen3-Coder-480B-A35B-Instruct [23] (Qwen3-C-480B in tables) and GLM-5-FP8 [28], under the OpenHands [18] scaffold with a 100-iteration ReAct budget. The P2T forward phase uses a sliding window of $n = 10$ steps with overlap $k = 5$. We fine-tune two student backbones, Qwen2.5-Coder-14B/32B-Instruct [3] (Qwen2.5-C-14B/32B in tables). Two prior recipes provide the baselines: *Test-pass rejection sampling* (SWE-Gym) [11], which keeps whole rollouts whose final patch passes $\mathcal{T}_i$; and *SWE-Lego* [16], which masks the SFT loss on assistant turns followed by a tool-error observation.

Evaluation. We evaluate on SWE-bench Verified (500 instances) and SWE-bench Lite (300) [5] under the same OpenHands scaffold and 100-iteration budget. Following the bi-objective target of Sec. 3.3, we report two metrics per (student, teacher) cell: *effectiveness* as resolve rate (Pass@1, ↑) under a single greedy rollout per instance; and *efficiency* as average per-instance inference cost in US$ (↓), metering prompt and completion tokens at official Alibaba Cloud Model Studio list prices for Qwen2.5-Coder-14B/32B-Instruct.[3] Cost is averaged over the full evaluation set, capturing the inference burden a downstream user incurs whether or not the rollout resolves the issue. Full SFT hyperparameters, parallelism, and context-window extension are deferred to App. J.

[3]Token rates from `https://www.alibabacloud.com/help/en/model-studio/model-pricing`; the same rate is applied to all conditions, so cost differences reflect only trajectory length.

Table 1: **Main evaluation results on SWE-bench Verified and SWE-bench Lite.** For each (student, teacher) pair we report two metrics: *effectiveness* as resolve rate (Pass@1, %, ↑) and *efficiency* as average per-instance inference cost (Cost, US$, ↓). **Bold** denotes the best result in each row; **(+X.X)** reports the absolute Pass@1 gain over the test-pass rejection-sampling baseline, **(−X.X)** the absolute cost reduction. Each cell is a single training run; three-seed results are reported in App. L.

Student	Teacher (Curator)	Metric	SWE-bench Verified				SWE-bench Lite			
			Baselines		Ours		Baselines		Ours	
			Test-Pass RS	SWE-Lego	Size-Matched	Full	Test-Pass RS	SWE-Lego	Size-Matched	Full
Qwen2.5-C-32B	Qwen3-C-480B	Pass@1 (%) ↑	39.6	40.6 (+1.0)	42.4 (+2.8)	**50.4** (+10.8)	28.7	28.7 (+0.0)	29.3 (+0.6)	**36.0** (+7.3)
		Cost ($) ↓	0.92	0.95 (+0.03)	0.85 (−0.07)	**0.78** (−0.14)	0.93	0.93	0.88 (−0.05)	**0.80** (−0.13)
	GLM-5-FP8	Pass@1 (%) ↑	38.4	38.8 (+0.4)	39.2 (+0.8)	**49.0** (+10.6)	31.6	32.3 (+0.7)	33.6 (+2.0)	**38.6** (+7.0)
		Cost ($) ↓	0.94	0.93 (−0.01)	0.89 (−0.05)	**0.81** (−0.13)	0.92	0.95 (+0.03)	0.85 (−0.07)	**0.78** (−0.14)
Qwen2.5-C-14B	Qwen3-C-480B	Pass@1 (%) ↑	36.0	36.2 (+0.2)	37.6 (+1.6)	**43.2** (+7.2)	22.0	22.7 (+0.7)	23.3 (+1.3)	**30.0** (+8.0)
		Cost ($) ↓	0.92	0.93 (+0.01)	0.87 (−0.05)	**0.78** (−0.14)	0.94	0.95 (+0.01)	0.90 (−0.04)	**0.83** (−0.11)
	GLM-5-FP8	Pass@1 (%) ↑	34.8	35.4 (+0.6)	36.6 (+1.8)	**42.8** (+8.0)	24.3	25.3 (+1.0)	26.3 (+2.0)	**32.0** (+7.7)
		Cost ($) ↓	0.93	0.94 (+0.01)	0.86 (−0.07)	**0.80** (−0.13)	0.93	0.92 (−0.01)	0.89 (−0.04)	**0.80** (−0.13)

5.2 Overall Effectiveness and Efficiency

We compare four trajectory-construction recipes under identical teacher, scaffold, and student-training pipelines: (i) the test-pass rejection-sampling baseline of SWE-Gym, (ii) the SWE-Lego process-level error-masking baseline, (iii) *P2T (size-matched)*, in which we randomly subsample our curated trajectories down to the size of the rejection-sampled pool to control for data scale, and (iv) *P2T (full)*, which uses every trajectory we curate from the 1.8k-instance training pool. Following the bi-objective trajectory target of Sec. 3.3, Table 1 reports both axes for each (student, teacher) pair: *effectiveness* as Pass@1 resolve rate (higher is better) and *efficiency* as average per-instance inference cost in US$ (lower is better), on SWE-bench Verified and SWE-bench Lite for two student backbones (Qwen2.5-C-14B/32B) and two teacher curators (Qwen3-C-480B and GLM-5-FP8).

Results. Across every (student, teacher, benchmark) cell of Table 1, P2T (full) is simultaneously more effective and more efficient than both baselines, lifting Pass@1 by up to $+10.8$ points (Qwen2.5-C-32B under Qwen3-C-480B on Verified) while cutting per-instance cost by $\sim 15\%$; SWE-Lego's tool-error masking yields at most marginal Pass@1 gains and never reduces cost, since it relabels the same rejection-sampled rollouts without changing trajectory length. Two controls explain where the lift comes from. First, the size-matched configuration already beats both baselines on both axes ($+2.8$ Pass@1 and $-\$0.07$ on Verified for the 32B/Qwen3-C-480B cell), so the gain is per-trajectory *quality*, not data scale. Second, moving to P2T (full) adds another $+8.0$ Pass@1 *while still lowering* cost; because the additional trajectories come from instances on which a blinded teacher rollout would have failed, this margin is supervision recovered from hard issues that rejection sampling silently discards, and the simultaneous cost drop rules out a verbosity confound. A complementary *compute-matched* control (App. K) further confirms that redirecting P2T's curation GPU-hours into $4\times$ additional teacher rollouts does not close the gap, isolating the gain from raw compute as well as from data scale. The gains are also robust to training randomness: repeating SFT with three independent seeds for the Qwen3-C-480B curator (App. L, Table 7), P2T (full) improves mean Pass@1 over Test-Pass RS by $+10.4$/$+7.1$ points (32B) and $+5.8$/$+7.2$ points (14B) on Verified/Lite with seed standard deviations of at most 1.5 points, and lowers cost in every setting. On the harder Python subset of SWE-bench Pro [2], P2T still lifts the 32B student from 8.3% to 10.5% Pass@1 (App. M). The lift is robust: under the weaker GLM-5-FP8 teacher absolute Pass@1 gains are within 0.2 pts and cost reductions within \$0.01 of those under Qwen3-C-480B, indicating the privileged-information factorization, not raw teacher capability, drives the improvement, and the same pattern transfers from the 32B to the 14B student.

5.3 Trajectory Quality

The improvements in §5.2 establish that P2T jointly raises resolve rate and lowers inference cost, but they do not yet explain *where* the savings come from. We trace the joint lift to a single mechanism: curated trajectories are shorter and less redundant *while covering more of the fix-relevant facts* the issue presupposes, and the property is induced both in the supervision data and in the rollouts the trained student emits. Below we report the population-level shifts in length and redundancy; an

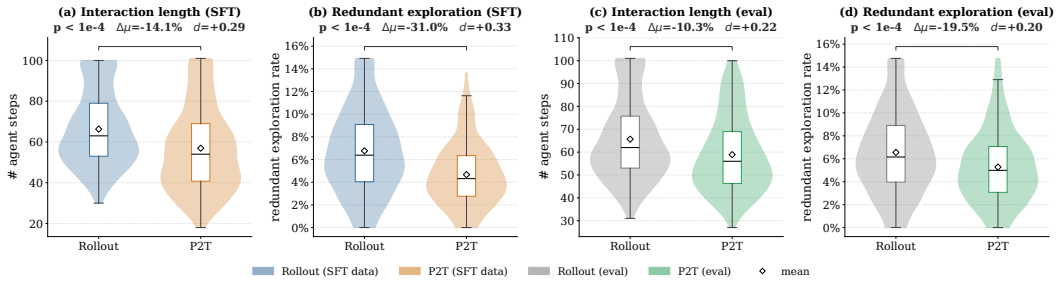


Figure 2: **Effect of P2T on trajectory quality, traced from supervision (a, b) to evaluation (c, d).** Each panel reports the paired Mann–Whitney p-value, the relative shift in mean ($\Delta\mu$), and Cliff's δ; diamonds mark means.

end-to-end worked example on `getmoto/moto` #6041, tracing both phases of P2T on this instance and contrasting the curated trajectory with the blinded rollout, is deferred to App. I (Fig. 7) for space.

Quantitative effect of curation. We measure two metrics: *interaction length* (number of agent steps) and *redundant exploration* (fraction of file-viewing actions whose visible range is fully covered by an earlier view in the same trajectory). Every comparison is restricted to instances that rollout resolve, and is measured at two pipeline stages: an *SFT-data view* pairing each rejection-sampled rollout with its P2T-curated counterpart on the same SWE-Gym instance under the GLM-5-FP8 teacher (Fig. 2 a,b), and a *student-eval view* comparing rollouts of the Qwen2.5-C-32B student trained on each supervision source on SWE-bench Verified (Fig. 2 c,d). At the SFT stage, P2T trajectories are -14.1% shorter ($66.3 \to 57$ steps) and -31.0% less redundant ($6.7\% \to 4.7\%$); the heavy upper tail of rollouts that exhaust the 100-iteration budget largely disappears (Fig. 2 a). Both shifts are paired-Mann–Whitney significant ($p < 10^{-4}$, $\delta = +0.29/+0.33$) and align with the design: the receding-horizon rule commits the shortest segment that clears the local effectiveness floor, while the groundedness critic suppresses the speculative re-inspection loops typical of unguided rollouts. Crucially, the same shifts reappear at evaluation time, where neither the curator nor any oracle is in the loop: the P2T-trained student emits rollouts that are -10.3% shorter and -19.5% less redundant than its rejection-sampled counterpart ($p < 10^{-4}$, $\delta \approx 0.20$). Behavioral cloning therefore transmits the structural property rather than merely the token-level distribution, which is the mechanism behind the $-\$0.13$–$\$0.14$ inference-cost gap in Table 1.

5.4 Component Ablation

We isolate the two design choices that the bi-objective program hinges on: the groundedness check that gates leakage on the effectiveness side, and the shortest-above-floor commit rule that controls length on the efficiency side. Each is removed in turn from the full pipeline; all other settings (Qwen3-C-480B teacher, OpenHands scaffold, 1.8k-instance pool, SFT recipe) are held fixed.

Effectiveness: groundedness check. Without the groundedness check, any frontier-advancing edit is committed verbatim, so curated trajectories may reference entities or claims the visible prefix does not yet support. On Qwen2.5-C-32B, removing it drops Pass@1 from 50.4% to 43.2% (-7.2 pts), with a similar -8.2 pt drop on Qwen2.5-C-14B. The student internalizes the same unjustified leaps at test time, exactly the failure mode the check was designed to block.

Efficiency: shortest-above-floor commit. Replacing the shortest-above-floor rule with a uniform-random pick from the candidate pool of equally effective segments removes the only term that pressures the trajectory to be short. Curated trajectories grow accordingly: average step count rises from 72.5 to 77.8 ($+7.3\%$) and per-trajectory token length from 64k to 72k ($+12.5\%$). Pass@1 also slips from 50.4% to 48.8% on Qwen2.5-C-32B, indicating that the shorter trajectories are not just cheaper but carry less imitation noise: behavioral cloning amplifies the lower-information steps that the rule would have pruned. Together, the two ablations show that the bi-objective rule is load-bearing on both axes simultaneously rather than trading them off.

Is $G^\star$ doing the work? A value-of-information study (App. C) confirms that the gains track $G^\star$ itself, not the forward-phase scaffolding: progressively disclosing $G^\star$ to a blinded reference solver lifts its Pass@1 from 29–35% to 94–97% under both teachers, and incidental $G^\star$ coverage of unguided

rollouts correlates strongly with success ($r \approx 0.36$). Both indicate that $G^\star$ encodes substantive prerequisites rather than post-hoc narration of $p^\star$.

6 Conclusion

We presented P2T, a framework that converts a reference patch into per-step process supervision for software-engineering agents without ever exposing the patch in the trajectory shown to the student. The pipeline factorizes curation into a reverse decomposition that distills a sufficient yet non-leaky prerequisite graph $G^\star$ from $p^\star$, and a forward grounded realization that uncovers $G^\star$ through ordinary tool calls under a hybrid groundedness critic and a surprisal trust region. On SWE-bench Verified, training on 1.7k curated SWE-Gym trajectories improves Qwen2.5-Coder-32B/14B-Instruct by $+10.8$/$+7.2$ points Pass@1 over an outcome-filtered baseline, with consistent gains on SWE-bench Lite and across two structurally different teacher curators. Controlled subsampling, a value-of-information study with progressive disclosure of $G^\star$, an observational coverage analysis, a quantitative trajectory-quality comparison, and a component ablation jointly indicate that the gains are attributable to the prerequisite-graph factorization itself rather than to data scale or any single safeguard. We discuss limitations and outlook in App. P.

Acknowledgments and Disclosure of Funding

This research is supported in part by the National Natural Science Foundation of China (62572300), the Minister of Education, Singapore (MOE-T2EP20124-0017, MOET32020-0004), the National Research Foundation, Singapore and the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN).

References

[1] Aleithan, R., Xue, H., Mohajer, M. M., Nnorom, E., Uddin, G., and Wang, S. SWE-bench+: Enhanced coding benchmark for LLMs. *arXiv preprint arXiv:2410.06992*, 2024.

[2] Deng, X., Da, J., Pan, E., He, Y. Y., Ide, C., Garg, K., Lauffer, N., Park, A., Pasari, N., Rane, C., et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

[3] Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Lu, K., et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[4] Jain, N., Singh, J., Shetty, M., Zheng, L., Sen, K., and Stoica, I. R2E-gym: Procedural environments and hybrid verifiers for scaling open-weights SWE agents. *arXiv preprint arXiv:2504.07164*, 2025.

[5] Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. R. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

[6] Kang, M., Chen, W.-N., Han, D., Inan, H. A., Wutschitz, L., Chen, Y., Sim, R., and Rajmohan, S. ACON: Optimizing context compression for long-horizon LLM agents. *arXiv preprint arXiv:2510.00615*, 2025.

[7] Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024.

[8] Ma, M., Liu, R., Lin, Y., Huang, Z., and Dong, J. S. TrainRef: Curating data with label distribution and minimal reference for accurate prediction and reliable confidence. In *The Fourteenth International Conference on Learning Representations*, 2026.

[9] Miettinen, K. *Nonlinear Multiobjective Optimization*, volume 12 of *International Series in Operations Research & Management Science*. Springer, 1999.

[10] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022.

[11] Pan, J., Wang, X., Neubig, G., Jaitly, N., Ji, H., Suhr, A., and Zhang, Y. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

[12] Peng, B., Quesnelle, J., Fan, H., and Shippole, E. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

[13] Qi, Z., Long, F., Achour, S., and Rinard, M. C. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *Proceedings of the 24th International Symposium on Software Testing and Analysis*, pp. 24–36. ACM, 2015. doi: 10.1145/2771783.2771791.

[14] Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635. PMLR, 2011.

[15] Smith, E. K., Barr, E. T., Le Goues, C., and Brun, Y. Is the cure worse than the disease? overfitting in automated program repair. In *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 532–543. ACM, 2015. doi: 10.1145/2786805.2786825.

[16] Tao, C., Chen, J., Jiang, Y., Kou, K., Wang, S., Wang, R., Li, X., Yang, S., Du, Y., Dai, J., et al. Swe-lego: Pushing the limits of supervised fine-tuning for software issue resolving. *arXiv preprint arXiv:2601.01426*, 2026.

[17] Vapnik, V. and Vashist, A. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5–6):544–557, 2009. doi: 10.1016/j.neunet.2009.06.042.

[18] Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

[19] Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., Tran, H. H., Li, F., Ma, R., Zheng, M., Qian, B., Shao, Y., Muennighoff, N., Zhang, Y., Hui, B., Lin, J., Brennan, R., Peng, H., Ji, H., and Neubig, G. OpenHands: An open platform for AI software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2025.

[20] Wang, Y., Pradel, M., and Liu, Z. Are" solved issues" in swe-bench really solved correctly? an empirical study. *arXiv preprint arXiv:2503.15223*, 2025.

[21] Xia, C. S., Deng, Y., Dunn, S., and Zhang, L. Agentless: Demystifying LLM-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

[22] Xiao, Y.-A., Gao, P., Peng, C., and Xiong, Y. Reducing cost of LLM agents with trajectory reduction. In *Proceedings of the ACM International Conference on the Foundations of Software Engineering (FSE)*, 2026. doi: 10.1145/3797084. arXiv:2509.23586.

[23] Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[24] Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K. R., and Press, O. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024. URL `https://arxiv.org/abs/2405.15793`.

[25] Yang, J., Lieret, K., Jimenez, C. E., Wettig, A., Khandpur, K., Zhang, Y., Hui, B., Press, O., Schmidt, L., and Yang, D. SWE-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025.

[26] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023.

[27] Zan, D., Huang, Z., Liu, W., Chen, H., Zhang, L., Xin, S., Chen, L., Liu, Q., Zhong, X., Li, A., et al. Multi-swe-bench: A multilingual benchmark for issue resolving. *arXiv preprint arXiv:2504.02605*, 2025.

[28] Zeng, A., Lv, X., Hou, Z., Du, Z., Zheng, Q., Chen, B., Yin, D., Ge, C., Huang, C., Xie, C., et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

[29] Zeng, L., Li, Y., Xiao, Y., Li, C., Liu, C. Y., Yan, R., Wei, T., He, J., Song, X., et al. Skywork-SWE: Unveiling data scaling laws for software engineering in LLMs. *arXiv preprint arXiv:2506.19290*, 2025.

[30] Zhang, L., He, S., Zhang, C., Kang, Y., Li, B., Xie, C., Wang, J., Wang, M., Huang, Y., Fu, S., et al. Swe-bench goes live! *arXiv preprint arXiv:2505.23419*, 2025.

[31] Zhang, Y., Ruan, H., Fan, Z., and Roychoudhury, A. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024.

[32] Zhao, Y., Huang, J., Hu, J., Wang, X., Mao, Y., Zhang, D., Jiang, Z., Wu, Z., Ai, B., Wang, A., Zhou, W., and Chen, Y. Swift:a scalable lightweight infrastructure for fine-tuning, 2024. URL `https://arxiv.org/abs/2408.05517`.

Appendix

A Technical appendices and supplementary material **14**

B Prerequisite-graph node distribution **14**

C Value of information in the prerequisite graph **14**

D Reverse-phase agent prompts and implementation details **16**

E Segment-wise commits and the trajectory-level objective **17**

F Candidate generation in detail **18**

G Full curation algorithm **19**

H Forward-phase agent prompts and implementation details **19**

I End-to-end worked example **22**

I.1 Issue x and oracle patch $p^\star$ 22

I.2 Phase 1 — Process graph distillation 22

I.3 Phase 2 — Receding-horizon trajectory realization 23

J Training details **26**

K Compute-matched rejection-sampling baseline **26**

L Multi-seed robustness of the main results **27**

M Evaluation on SWE-bench Pro **28**

N Pipeline yield by stage **29**

O Human audit of residual leakage **29**

P Limitations and outlook **30**

A Technical appendices and supplementary material

This appendix provides material complementary to the main paper: prerequisite-graph statistics (App. B), a value-of-information study for $G^\star$ (App. C), reverse-phase agent prompts (App. D), the segment-wise commit analysis (App. E), candidate generation in detail (App. F), the full curation algorithm (App. G), forward-phase agent prompts (App. H), an end-to-end worked example (App. I), training details (App. J), the compute-matched RS baseline (App. K), three-seed robustness of the main results (App. L), evaluation on SWE-bench Pro (App. M), per-stage pipeline yield (App. N), a human audit of residual leakage (App. O), and limitations and outlook (App. P).

B Prerequisite-graph node distribution

Figure 3 characterizes the prerequisite graphs $\{G_i^\star\}$ produced by the reverse phase on the $N = 1{,}815$ SWE-Gym training instances used to curate P2T trajectories, comprising $33{,}106$ in-scope nodes in total. Panel (a) shows that the aggregate node population is dominated by *static facts* (66.2%, e.g., file/symbol locations and type signatures readable from the repo at h), followed by *dynamic facts* (16.7%, observations that require executing code), and the three artifact categories—reproduction, analysis, and fix plan—each contributing roughly 5–6%. Panel (b) reports the per-instance mean count alongside the fraction of instances that contain at least one node of each category: every instance has at least one static fact and a fix plan (100% coverage), reproduction and analysis artifacts appear in essentially every instance as well, while dynamic facts have lower per-instance mass (3.0 nodes on average) but appear whenever the issue's behavior is meaningfully runtime-dependent. Panel (c) gives the distribution of total in-scope graph size: the mode is concentrated around 15–20 nodes (median 18, mean 18.2), with a thin right tail extending past 25. Panels (d,e) examine variation across instances. The stacked composition in (d), with instances sorted by total graph size, shows that the static-fact share grows roughly linearly with graph size, while artifact counts (≤ 2 per category in nearly all instances) are remarkably stable; the per-category spreads in (e) confirm this—artifact categories are tightly concentrated near their median, whereas static and dynamic facts carry essentially all of the cross-instance variance.

Three consequences for our method follow. First, because static facts dominate, the bulk of the curator's frontier-advancement work consists of grounded read-only inspection actions (file reads, symbol lookups, ripgrep), which are cheap and naturally on-policy for a non-privileged solver. Second, the small but stable artifact budget per instance (~ 1–3 nodes each for reproduction/analysis/plan) bounds the number of synthesized intermediate steps the forward phase has to inject, keeping curated trajectories close in length to organic rollouts. Third, the long tail in panel (c) identifies a sub-population of instances with large graphs (> 25 nodes) where the gap between curated and rejection-sampled trajectories is largest, since these are precisely the instances on which a single greedy blinded rollout is least likely to incidentally cover the required prerequisites (cf. App. C).

C Value of information in the prerequisite graph

The improvements in Sec. 5.2 establish that P2T trajectories help *a student*, but they do not on their own show that the prerequisite graph $G^\star$ itself carries the right information; one might worry that the gains come entirely from the forward-phase machinery (segment-level bi-objective optimization selection, groundedness gating) and that $G^\star$ is little more than a post-hoc rationalization of $p^\star$. This appendix tests the alternative directly through two complementary protocols. First, an *interventional* disclosure study progressively reveals nodes of $G^\star$ to a blinded reference solver and measures the resulting end-to-end resolve rate. Second, an *observational* coverage study verifies that, even when $G^\star$ is never exposed, blinded rollouts whose incidental coverage of $G^\star$ is highest are also the ones most likely to succeed. Both directions converge on the same conclusion.

Protocol (interventional disclosure). We define a nested chain of information bundles $B^{(0)} \subset B^{(1)} \subset B^{(2)} \subset B^{(3)} \subset B^{(4)}$, each augmenting the visible prefix at the start of an episode:

- $B^{(0)}$: issue x only;
- $B^{(1)}$: $B^{(0)}+$ all facts in $V_F^\star$ (context);

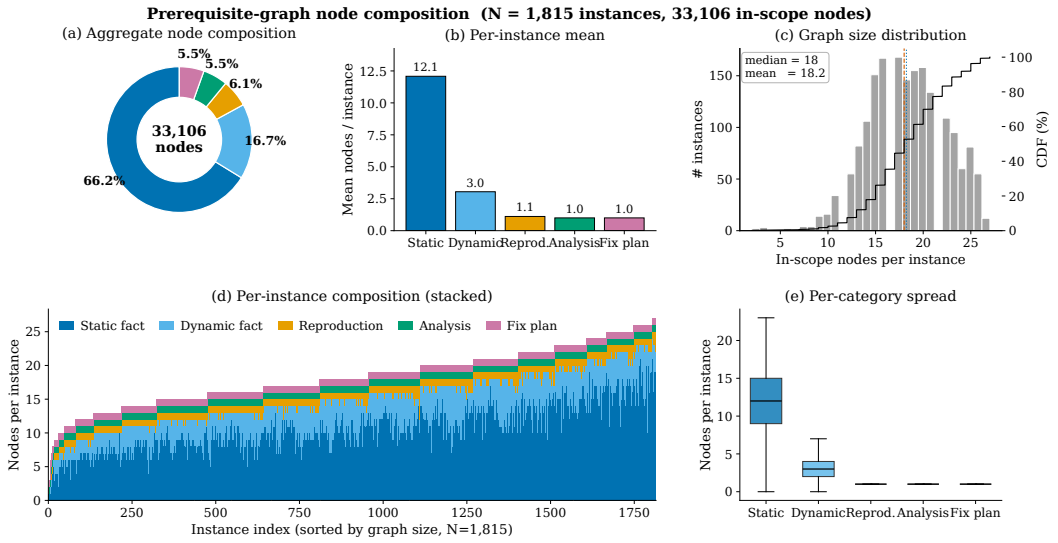


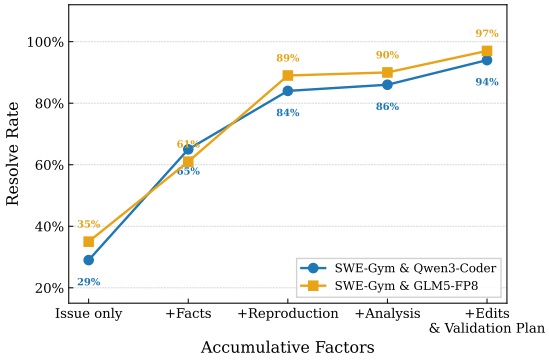
Figure 3: **Composition of the prerequisite graphs $\{G_i^\star\}$ across the $N = 1{,}815$ SWE-Gym training instances** ($33{,}106$ **in-scope nodes). (a)** Aggregate node-type breakdown: static facts dominate (66.2%), with dynamic facts (16.7%) and the three artifact categories (reproduction, analysis, fix plan) at 5–6% each. **(b)** Per-instance mean count by category (bars) and fraction of instances containing at least one node of that category (right axis): every instance has at least one static fact and fix plan; dynamic facts are sparser on average but still present in most instances. **(c)** Distribution of total in-scope graph size across instances (median 18, mean 18.2) and its CDF. **(d)** Per-instance stacked composition with instances sorted by graph size, showing that static-fact share grows with size while artifact counts remain stable. **(e)** Per-category spread: artifact categories are tightly concentrated, whereas static and dynamic facts account for nearly all of the cross-instance variance.

- $B^{(2)}$: $B^{(1)}+$ the reproduction-script artifact;
- $B^{(3)}$: $B^{(2)}+$ the root-cause analysis artifact;
- $B^{(4)}$: $B^{(3)}+$ the fix-plan artifact.

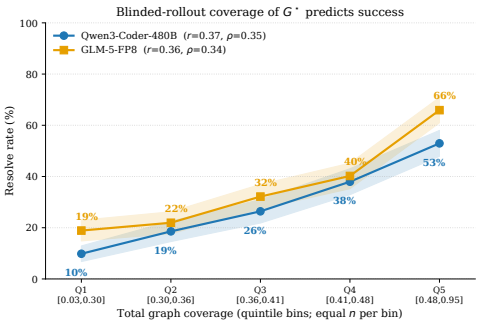
Crucially, $p^\star$ is never disclosed at any stage, so each bundle is something a non-privileged solver could in principle have constructed for itself. We restrict the study to the 1.8k training instances on which $p_i^\star$ passes $\mathcal{T}_i$ (the same pool used for our curated trajectories) so that variation in resolve rate is attributable to the bundle, not to broken evaluation environments. For each bundle and each teacher, the same teacher model is then used as the reference solver under the OpenHands scaffold with the standard 100-iteration budget, and we measure Pass@1 averaged over a single rollout per instance.

Results (interventional). Figure 4 shows that the resolve rate increases monotonically as more of $G^\star$ is revealed, under both teachers. From issue-only context, the reference solver resolves 29% (35%) of instances under the Qwen3-Coder-480B (GLM-5-FP8) curator. Adding the fact statements alone—no scripts, no plans—more than doubles this to 65% (61%). Revealing the reproduction script delivers the largest single jump, to 84% (89%); the root-cause analysis adds a further $+2$ ($+1$) points; and the fix plan plus validation stub take performance to 94% (97%). Every marginal addition is non-negative for both teachers.

Three observations follow. First, the magnitude of the lift from $B^{(0)}$ to $B^{(4)}$ in Fig. 4a—roughly $+65$ percentage points—is far too large to be explained away as cosmetic narration: the curated graph encodes substantive, action-relevant knowledge. Second, the bulk of the improvement comes from facts ($+\sim$30 pts) and the reproduction script ($+\sim$20 pts), confirming that the reverse phase's two staples—atomic fact distillation and concrete artifact scaffolding—are precisely the components that carry information for a non-privileged solver. Third, the consistency of the trend across two structurally different teachers indicates that the prerequisite-graph structure, rather than any teacher-specific stylistic bias, is what supplies the value of information.



(a) Progressive disclosure of $G^\star$ to a blinded solver.



(b) Observational coverage of $G^\star$ in blinded rollouts vs. resolve rate.

Figure 4: **Value of information in $G^\star$. (a)** Resolve rate of a blinded reference solver on the 1.8k training instances as elements of the prerequisite graph are progressively revealed; the oracle patch $p^\star$ is never exposed. Each addition contributes a non-negative marginal gain under both teachers, with facts and the reproduction script accounting for most of the lift. **(b)** When the graph is *not* disclosed, instances on which a blinded rollout incidentally covers more nodes of $G^\star$ are resolved at substantially higher rates: rollouts in the top coverage quintile succeed at 53–66%, vs. 10–19% in the bottom quintile, with a consistent positive correlation under both teachers ($r \approx 0.36$, $\rho \approx 0.35$).

Observational coverage predicts success without disclosure. A natural concern with the disclosure protocol is that prepending structured text to the prompt may help for reasons unrelated to the *content* of $G^\star$—e.g., framing or anchoring effects. We therefore complement Fig. 4a with an *observational* test in which the graph is never exposed to the solver. For each of the 1.8k instances, we take a fully blinded rollout under each teacher (issue only, no $G^\star$, no $p^\star$) and apply the same unlock criterion the curator uses during forward realization—a node is counted as covered when an unlocker action (or an equivalent action) is executed and the associated statement (or an equivalent concept) is encoded in the trajectory. The total coverage of an instance is then the fraction of nodes in $V_F^\star \cup V_A^\star$ that are covered. We bin instances into quintiles of equal size by this coverage and report the resolve rate per bin in Fig. 4b.

Resolve rate increases monotonically with coverage under both teachers, climbing from 10%/19% in the lowest-coverage quintile to 53%/66% in the highest, with Pearson $r \approx 0.36$ and Spearman $\rho \approx 0.35$. Because the solver is given no privileged information in this analysis, the correlation cannot be an artifact of prompt augmentation: it shows that the very prerequisites identified by reverse decomposition are, on the issues a blinded solver *does* resolve, the ones it tends to establish on its own. The two panels of Fig. 4 thus triangulate the same conclusion from opposite directions—disclosing $G^\star$ helps, and independently establishing $G^\star$ correlates with success—providing converging evidence that $G^\star$ tracks information genuinely required to solve the issue rather than post-hoc narration of $p^\star$, and justifying the use of $G^\star$ as the per-step effectiveness signal in Sec. 4.2.

D Reverse-phase agent prompts and implementation details

We instantiate the proposer–critic loop of Sec. 4.1 with two LLM agents that share a tool-mediated view of the repository (file-viewer, ripgrep, sandboxed `python`/`pytest`) and exchange JSON node sets. Both agents see the issue d_i, the repository R_i, the reference patch $p_i^\star$, and the test suite $\mathcal{T}_i$; only the proposer is allowed to introduce new nodes, and only the critic is allowed to delete or revise them. The DAG-organization step is a deterministic post-pass: an edge is drawn from u to v iff v's unlocker action references an entity (file, symbol, line range, runtime value) first surfaced by u's unlocker observation, or v is an artifact whose type strictly follows u's in the canonical order repro $\to$ analysis $\to$ plan $\to$ edit $\to$ validation.

Unlocker taxonomy. Each node carries an *unlocker* $u_v = (\text{ACTION}, \text{OBSERVATION})$ drawn from a fixed taxonomy: (i) VIEW$\langle$file, lines$\rangle$ for static reads; (ii) VIEW `problem_statement` for issue-derived facts; (iii) BASH$\langle$command$\rangle$ for grep, runtime probing, or test execution (dynamic facts); (iv) CREATE$\langle$path, content$\rangle$ and STR_REPLACE$\langle$file, old, new$\rangle$ for reproduction scripts and edits;

(v) THINK⟨text⟩ for analysis and fix-plan artifacts. Every action string must be copy-pasteable; abbreviations such as ... are rejected by the critic on sight.

Stopping criterion. Writing $V^{(t)}$ for the node set after critic round t, the loop terminates when (a) $V^{(t)} = V^{(t-1)}$ (set-level fixed point under hashed canonical statements), or (b) the proposer returns $\Delta V = \varnothing$ for two consecutive rounds, or (c) a hard cap of $T_{\max} = 6$ rounds is reached. In our runs, 96% of instances converge by round 3.

LLM and decoding. Both agents are run on the same teacher backbone used for blinded rollouts (Qwen3-Coder-480B or GLM-5-FP8) with temperature 0.4, top-p 0.95, max output 8k tokens, JSON-mode constrained decoding, and a 40-step tool-call budget per round. Outputs are validated against a JSON schema; on schema violation the agent is re-prompted up to twice with the validator error before the round is dropped.

The two prompt templates below are *abstracted* versions of the production prompts: we preserve the role, inputs, hard rules, and output schema verbatim but elide engineering boilerplate (tool-API specifications, output-format examples, retry instructions, and repository-specific stop-lists) for readability.

Proposer prompt (reverse phase, Sec. 4.1)

Role. You are the *Proposer*. Working backwards from a known-correct reference patch $p^\star$, your job is to enumerate the contextual facts and solution milestones a non-privileged developer would need to discover before $p^\star$ becomes derivable. You are NOT producing a description of $p^\star$ — you are producing the prerequisite knowledge that motivates $p^\star$.
Inputs. Issue d_i; repository R_i (read/grep/exec via tools); reference patch $p_i^\star$; failing-to-passing tests; current node set $V^{(t)}$ with critic feedback $\phi^{(t)}$.
Task. Emit $\Delta V^{(t+1)}$: a JSON list of new candidate nodes that close the largest remaining gap to $p^\star$. Each node has `id`, `node_type` $\in$ {fact, reproduce_script, issue_analysis, fix_plan, code_edit, validation}, a one-claim `statement`, and a fully-specified `unlocker`. Cover *every* logically distinct change in $p^\star$: imports, registration entries, exception classes, parameter additions, helper reuse, schema/template updates.
Hard rules. (R1) One claim per fact; split compounds. (R2) The unlocker action is restricted to the taxonomy above; no `[view] golden patch`, `[view] hints`, or any reference to $p^\star$. (R3) Action strings are complete and replayable — no ..., no prose placeholders. (R4) Observations are the actual tool output you obtained when you executed the action — do not paraphrase from $p^\star$. (R5) For every `reproduce_script` and `validation` node, you must execute the script in the sandbox and record `output_before_fix`/`actual_output`; nodes without recorded execution are rejected. (R6) For every `code_edit`, `old_str` must match the current pre-fix source byte-for-byte.
Investigation protocol. (i) Index every distinct change in $p^\star$. (ii) For each change, trace the call chain backwards to the user-facing symptom and forwards to the user-facing effect. (iii) Search the repository for *analogous implementations* and cite at least one concrete location (file + line range) when $p^\star$ follows an existing pattern. (iv) Probe runtime behavior with `python -c` or `pytest` when types, dispatch, or values are not statically obvious. (v) Narrow the bug: test the failing case AND the working case to confirm the fix scope is minimal. (vi) Apply the critic feedback $\phi^{(t)}$ and address each cited gap.
Output. A JSON object {instance_id, nodes : [...]} written via `create_file` to the designated path. Nothing else.

Critic prompt (reverse phase, Sec. 4.1)

Role. You are the *Critic*. Your job is to enforce desideratum (ii), *non-leakage*: every surviving node's unlocker must be conceivable from the issue, the repository, and the node's predecessors — never from $p^\star$. You also enforce minimality and discoverability.
Inputs. d_i, R_i, $p_i^\star$, $V^{(t)} \cup \Delta V$ (the proposer's latest output).
Per-node verdict. For each candidate v, assign exactly one of: **keep** (passes all checks); **prune** (fails at least one hard check); **revise** (salvageable with the noted edit). Output a JSON list with `id`, `verdict`, `reasons[]`, and (for revise) `patched_node`.
Hard checks (any failure ⇒ prune). (H1) PATCH LEAKAGE. The statement describes what $p^\star$ does (e.g. "the patch adds X", "the fix is to call Y") rather than a property of the pre-fix codebase or runtime. (H2) FORBIDDEN UNLOCKER. Action references $p^\star$, the test patch, or hidden hints; uses a tool outside the taxonomy; or contains an abbreviation that prevents replay. (H3) UNVERIFIED OBSERVATION. The recorded `observation` cannot be reproduced by re-executing `action` on the pre-fix repository. (H4) PREMATURE ARTIFACT. A `code_edit` node appears without an upstream `fix_plan`; a `fix_plan` appears without an upstream `issue_analysis`; a `validation` appears without the `code_edit` it validates. (H5) COMPOUND CLAIM. A fact bundles two or more distinct propositions. (H6) EDIT DRIFT. A `code_edit` node's `old_str` does not match the current source byte-for-byte, or the union of all edits does not reproduce $p^\star$.
Soft checks (issue `revise`). (S1) Two facts state the same claim with different evidence ⇒ merge, keeping the stronger evidence. (S2) An action is replayable but unnecessarily wide (e.g. `view 1-500`) when 10 lines suffice ⇒ tighten the line range. (S3) A static fact whose statement obviously requires execution to verify ⇒ re-tag as dynamic and provide a probe.
Feedback to proposer. For every cited gap, append a one-sentence ϕ-entry naming (a) the missing fact category, and (b) a concrete next investigation (e.g. "add a fact identifying the analogous handler in the same file"). The feedback is consumed verbatim by the next proposer round.
Output. A JSON object {instance_id, verdicts : [...], feedback : [...]} written via `create_file`. The next proposer round receives $V^{(t+1)} = V^{(t)} \cup \{v : \text{verdict}(v) = \text{keep}\} \cup \{\text{patched}(v) : \text{verdict}(v) = \text{revise}\}$ together with the feedback list.

E Segment-wise commits and the trajectory-level objective

Sec. 4.2 commits trajectories one segment at a time: at each window t it forms a candidate pool $\mathcal{S}_t$, restricts to the locally non-dominated set $\mathcal{P}_t = \mathrm{ND}(\mathcal{S}_t)$ under $(\mathrm{Eff}_t, \mathrm{Len}_t)$, and commits $s_t^\star =$

$\arg\min_{s \in \mathcal{P}_t} \mathrm{Len}_t(s)$ s.t. $\mathrm{Eff}_t(s; G_i^\star) \geq \eta_t$. Section 3.3, by contrast, formulates a *trajectory-level* rule: among trajectories satisfying $\mathrm{Eff}_i(\tau) \geq \eta_i$, pick the shortest. We sketch why the segment-wise commits realize the trajectory-level rule on the family of trajectories that the per-window pools induce.

Setup. A trajectory built by the procedure decomposes into disjoint windows, $\tau = s_1 \oplus \cdots \oplus s_W$, where $\oplus$ denotes concatenation along the prefix. Both objectives are additive over this partition:

$$\mathrm{Len}_i(\tau) \;=\; \sum_{t=1}^{W} \mathrm{Len}_t(s_t), \qquad \mathrm{Eff}_i(\tau) \;=\; \sum_{t=1}^{W} \mathrm{Eff}_t(s_t),$$

where Len is response-token mass (or step count) and $\mathrm{Eff}_t(s_t) = \sum_{\tau \in s_t} \mathrm{Prog}_\tau$ on segments that pass the groundedness gate. Additivity holds because windows are non-overlapping and Prog_τ is computed against the realized-node set $U_{\tau-1}$ at the step's own prefix, with the leakage rejection inside Prog_τ and the groundedness gate Ground_t operating on disjoint scopes within a segment (Sec. 4.2).

Floor calibration. Let the per-window floors be calibrated so that $\sum_{t=1}^{W} \eta_t = \eta_i$. In practice we set a constant $\eta_t = \eta_i / W^\star$ for an expected horizon $W^\star$ and absorb stochasticity in the fallback rule of Sec. 4.2 (commit the maximum-effectiveness segment when no candidate clears the floor); the proof statement below pertains to the deterministic case in which the floor is met at every window.

Greedy feasible set. Conditioned on the prefix produced by previous commits, each window induces a candidate pool $\mathcal{S}_t$ and an admissible subset $\mathcal{F}_t = \{s \in \mathcal{P}_t \mid \mathrm{Eff}_t(s) \geq \eta_t\}$. The *greedy feasible family* is

$$\mathcal{T}_{\mathrm{greedy}} \;=\; \big\{\, s_1 \oplus \cdots \oplus s_W \;:\; s_t \in \mathcal{F}_t \text{ under prefix } s_1 \oplus \cdots \oplus s_{t-1} \,\big\},$$

i.e. all trajectories formable by picking one admissible segment per window from the pools the procedure actually encounters.

Claim. The trajectory $\tau^\star = s_1^\star \oplus \cdots \oplus s_W^\star$ returned by the segment-wise procedure satisfies

$$\mathrm{Eff}_i(\tau^\star) \;\geq\; \eta_i, \qquad \mathrm{Len}_i(\tau^\star) \;=\; \min_{\tau \in \mathcal{T}_{\mathrm{greedy}}} \mathrm{Len}_i(\tau),$$

and is therefore the shortest-above-floor trajectory in $\mathcal{T}_{\mathrm{greedy}}$.

Proof sketch. *(i) Floor.* For each window t, the commit rule enforces $\mathrm{Eff}_t(s_t^\star) \geq \eta_t$. Summing across windows and using additivity of Eff gives $\mathrm{Eff}_i(\tau^\star) = \sum_t \mathrm{Eff}_t(s_t^\star) \geq \sum_t \eta_t = \eta_i$. *(ii) Length-optimality.* Fix any $\tau = s_1 \oplus \cdots \oplus s_W \in \mathcal{T}_{\mathrm{greedy}}$. Pointwise minimality of $s_t^\star$ within $\mathcal{F}_t$ gives $\mathrm{Len}_t(s_t^\star) \leq \mathrm{Len}_t(s_t)$ at every t; summing and using additivity of Len yields $\mathrm{Len}_i(\tau^\star) \leq \mathrm{Len}_i(\tau)$. Combined with (i), $\tau^\star$ is feasible and length-minimal in $\mathcal{T}_{\mathrm{greedy}}$, which is the trajectory-level shortest-above-floor rule restricted to that family.

Scope. The proof shows *no Pareto regret within* $\mathcal{T}_{\mathrm{greedy}}$, not global optimality over the full trajectory space $\mathcal{T}(\mathcal{I}_i)$ of Sec. 3.3: greedy commits change the prefix, and a different early commit could open downstream pools containing strictly better segments. Two choices control this gap. First, replanning (the receding-horizon $\delta < n$ execution in Sec. 4.2) shrinks the prefix that any single commit locks in, reducing the loss from greediness. Second, drawing seeds from π_{blind} and bounding curator intervention to one edit per segment keeps the per-window pools well-distributed across reasonable continuations, so the greedy family is a representative slice of the trajectory space rather than a degenerate corner of it. The empirical comparison against rejection sampling and the ablations in Sec. 5 measure how much this restriction costs in practice.

F Candidate generation in detail

This appendix expands the candidate-pool construction sketched in Sec. 4.2.

Blinded seeds. At window t we draw K length-n continuations

$$\tilde{s}_t^{(k,0)} \sim \pi_{\text{blind}}^{(n)}(\cdot \mid h_t), \qquad k = 1, \ldots, K,$$

each executed in a sandbox copy of the environment so that side effects on R_i (file edits, test runs, package installs) do not leak across candidates.

Single-mutation variants. For each seed $\tilde{s}_t^{(k,0)}$ the curator may additionally pick a position $t + j$ ($0 \le j < n$) and a target node $v \in \mathcal{A}_{t+j}(G_i^\star)$ available under the simulated prefix $h_{t+j}^{(k)}$, then draw a one-step replacement response

$$y'_{t+j} \sim q_{\text{mut}}(\cdot \mid h_{t+j}^{(k)},\, v)$$

that points the next action at v's unlocker. The sandbox is rolled back to step $t + j$, y'_{t+j} is executed, and the remaining $n - j$ steps of the suffix are regenerated from π_{blind} under the new prefix. Indexing the resulting variants by their target m, the window-t pool is

$$\mathcal{S}_t \;=\; \{\tilde{s}_t^{(k,0)}\}_{k=1}^{K} \;\cup\; \{s_t^{(k,m)}\}_{k,m}.$$

Why one edit per segment. Confining curator intervention to a single step per segment plays two roles. First, it isolates causality inside the local Pareto problem: every variant differs from some pure seed by exactly one localized rewrite, so any $(\text{Eff}_t, \text{Len}_t)$ gap between them can be attributed to that edit, which makes the per-segment selection rule of Sec. 4.2 well-posed and the groundedness gate cheap to evaluate (it inspects only y'_{t+j}). Second, it keeps each committed action a blinded action up to a single rewrite, bounding the trajectory's deviation from the student-facing distribution and ensuring the resulting tokens remain safe imitation targets under SFT.

G Full curation algorithm

Algorithm 1 gives the end-to-end pseudocode for P2T, combining the process graph distillation of Sec. 4.1 with the receding-horizon bi-objective trajectory realization of Sec. 4.2. The notation matches Sec. 4: $G_i^\star = (V_i, E_i)$ is the distilled process graph, U_t is the realized-node set, $\mathcal{A}_t$ the available frontier, Prog_τ the per-step progress score, Ground_t the binary groundedness gate, and (η_t, n, K, δ) the per-window floor, segment length, seed count, and commit horizon.

H Forward-phase agent prompts and implementation details

The forward phase (Sec. 4.2) instantiates four specialized components, each with a fixed prompt template: a *curator* that proposes single-step mutations toward an available frontier node; a symbolic *entity extractor* that implements the referential-integrity half of Ground_t; a neural *claim-grounding judge* that implements the entailment half of Ground_t; and a node-*establishment verifier* that decides $\text{Est}(v, h_t)$ for the per-step progress score Prog_t (Sec. 4.1). All four components see only the visible prefix h_t, the issue d_i, and (for the curator and verifier) the relevant graph nodes; none of them ever sees $p_i^\star$ or the test suite. The prompt panels below are *abstracted* versions of the production prompts — role, inputs, hard rules, and output schema are preserved verbatim, while tool-API specifications, output-format examples, and engineering boilerplate are elided for readability. The full production prompts will be released with the code. The curator additionally receives the available frontier $\mathcal{A}_{t+j}$ and the realized set U_{t+j} at the position it chooses to mutate.

Sliding-window and hyperparameter settings. We use a window length $n = 10$ ReAct steps, half-segment stride (commit horizon $\delta = 5$, so consecutive windows overlap by 5 steps), and $K = 2$ blinded seeds per window plus up to one mutation per seed (≤ 4 candidates per pool). The per-window floor is $\eta_t = \max(1, |\mathcal{A}_{t-1}|) \cdot 0.5$ (i.e., realize at least half the currently available frontier), with the fallback rule of Sec. 4.2 when no candidate clears the floor. Seeds are drawn from π_{blind} at temperature 0.6 (top-p 0.95, max output 2k tokens per step), with a per-seed sandbox snapshot rolled back after scoring. The curator and judge run on the same teacher backbone as the seeds at temperature 0.2; the entity extractor is a deterministic Python pass over the regex set listed below.

Algorithm 1 P2T: oracle-guided process graph distillation and trajectory realization

Require: Task instance $\mathcal{I}_i = (d_i, R_i, E_i, \mathcal{T}_i, p_i^\star)$; blinded solver π_{blind}; window length n; seeds per window K; commit horizon $\delta \le n$; per-window floor η_t
Ensure: Curated trajectory $\tau_i^\star$ admitted into $\mathcal{D}_{\text{ours}}$, or $\perp$

Phase 1: Process Graph Distillation (Sec. 4.1)

$V \leftarrow \varnothing$
repeat ▷ proposer–critic loop
  $\Delta V \leftarrow \text{Proposer}(d_i, R_i, V, p_i^\star)$ ▷ (i) sufficiency: close logical gap to $p_i^\star$
  $V \leftarrow \text{Critic}(V \cup \Delta V, d_i, R_i)$ ▷ (ii) non-leakage: prune nodes whose unlocker presupposes $p_i^\star$
until V stabilizes
$E \leftarrow \{(u, v) : u \text{ must be realized before } v\text{'s unlocker applies}\}$ ▷ (iii) feasible ordering
$G_i^\star \leftarrow (V, E)$

Phase 2: Receding-Horizon Bi-Objective Realization (Sec. 4.2)

$h \leftarrow d_i;\ U \leftarrow \varnothing;\ \tau \leftarrow ();\ t \leftarrow 0$
while agent has not call finish tool **do**
  $\mathcal{A} \leftarrow \{v \in V \setminus U : \text{Pred}_{G_i^\star}(v) \subseteq U\}$ ▷ available frontier
  $\mathcal{S}_t \leftarrow \varnothing$
  for $k = 1, \ldots, K$ **do** ▷ K blinded seeds, sandboxed
    $\tilde{s}^{(k,0)} \sim \pi_{\text{blind}}^{(n)}(\cdot \mid h)$; add to $\mathcal{S}_t$ with $\text{Ground} = 1$
    for each position $j \in \{0, \ldots, n-1\}$ and target $v \in \mathcal{A}_{t+j}(G_i^\star)$ under $\tilde{s}^{(k,0)}$ **do**
      Roll back sandbox to step $t + j$; draw $y'_{t+j} \sim q_{\text{mut}}(\cdot \mid h_{t+j}^{(k)}, v)$ ▷ single-mutation variant
      Execute y'_{t+j}; regenerate suffix from $\pi_{\text{blind}} \Rightarrow s^{(k,v)}$
      $\text{Ground}_t(s^{(k,v)}) \leftarrow \mathbb{1}[\text{Ents}(y'_{t+j}) \subseteq \text{Obs}(h_{t+j})] \cdot \text{Claim}(y'_{t+j}, h_{t+j})$
      Add $s^{(k,v)}$ to $\mathcal{S}_t$
    end for
  end for
  for each $s \in \mathcal{S}_t$ **do** ▷ score under $(\text{Eff}_t, \text{Len}_t)$
    Simulate s on h; for each step $\tau \in s$ compute $U_\tau, \mathcal{A}_{\tau-1}, \Delta_\tau = U_\tau \setminus U_{\tau-1}$
    $\text{Prog}_\tau \leftarrow \dfrac{|\Delta_\tau|}{\max(|\mathcal{A}_{\tau-1}|,\ 1)} \cdot \mathbb{1}\big[\ \Delta_\tau \subseteq \mathcal{A}_{\tau-1}\ \big]$ ▷ indicator hard-zeros leaky steps
    $\text{Eff}_t(s) \leftarrow \text{Ground}_t(s) \cdot \sum_{\tau \in s} \text{Prog}_\tau;\ \text{Len}_t(s) \leftarrow \sum_{y \in s} |y|$ (or $|s|$)
  end for
  $\mathcal{P}_t \leftarrow \text{ND}(\mathcal{S}_t)$ under $(\text{Eff}_t, \text{Len}_t)$; $\mathcal{F}_t \leftarrow \{s \in \mathcal{P}_t : \text{Eff}_t(s) \ge \eta_t\}$
  if $\mathcal{F}_t \neq \varnothing$ **then**
    $s_t^\star \leftarrow \arg\min_{s \in \mathcal{F}_t} \text{Len}_t(s)$ ▷ shortest-above-floor
  else
    $s_t^\star \leftarrow \arg\max_{s \in \mathcal{P}_t} \text{Eff}_t(s)$ ▷ fallback
  end if
  $s_t^{\text{commit}} \leftarrow$ first δ steps of $s_t^\star$
  Execute s_t^{commit} on the live environment; append (y, o) pairs to τ and h; update U
  $t \leftarrow t + \delta$
end while
$\hat{p} \leftarrow \text{patch}(\tau)$
if $\mathcal{T}_i(\hat{p}) = 1$ **then return** $\tau_i^\star \leftarrow \tau$
else return $\perp$
end if

Entity-extractor pattern set (symbolic check). $\mathrm{Ents}(y)$ and $\mathrm{Obs}(h)$ are defined as the union of strings matched by the following Python-regex family, applied to the assistant message and tool-call arguments of y (resp. to all observations and assistant messages in h):

Entity regex set (case-sensitive; flags re.MULTILINE)

```
FILE_PATH_REL  :=  (?:[\w.\-]+/)+[\w.\-]+\.(?:py|pyx|pyi|c|cpp|h|js|ts|json|yaml|yml|toml|cfg|md|txt|sh)
FILE_PATH_ABS  :=  /(?:workspace|testbed|opt|usr|home)/[\w./\-]+
DOTTED_MODULE  :=  (?:[a-z_][a-z0-9_]*\.)1,[a-zA-Z_][a-zA-Z0-9_]*
QUALIFIED_NAME  :=  \b[A-Z][A-Za-z0-9_]*(?:\.[A-Za-z_][A-Za-z0-9_]*)+\b
IDENTIFIER_DEF  :=  (?:def|class|async\ def)\s+([A-Za-z_][A-Za-z0-9_]*)
IDENTIFIER_REF  :=  \b[_a-zA-Z][_a-zA-Z0-9]{2,}\b  (filtered by stop-list of Python keywords + 200 most
common English words)
LINE_REF  :=  (?:line|lines|L)\s#?\d+(?:\s*[--]\s*\d+)?
ERROR_TYPE  :=  \b[A-Z][A-Za-z]*(?:Error|Exception|Warning)\b
SHELL_FLAG  :=  (?<=\s)-?[a-zA-Z][a-zA-Z_\-]+
NUMERIC_LITERAL  :=  \b\d{3,}\b  (e.g., issue numbers, large constants; small ints are excluded as
non-discriminative)
```

The symbolic check passes iff every match in y is also a match in h, modulo a path-normalization step that strips workspace prefixes and a trailing-suffix collapse for nested attribute access (`a.b.c` matches if either `a.b.c` or `b.c` appears in h). Single-token English identifiers shorter than three characters and members of the keyword stop-list are excluded from $\mathrm{Ents}(y)$ to avoid spurious failures. Empirically the symbolic gate fires on 13.4% of mutated candidates, and 91% of those are downstream rejected by the neural judge as well.

Curator prompt. Given a sandboxed seed $\tilde{s}_t^{(k,0)}$, the curator picks a position j and a target $v \in \mathcal{A}_{t+j}$, then writes a one-step replacement y'_{t+j} intended to advance toward v's unlocker. It does not receive $p^\star$; it sees only the issue, the visible prefix at position $t+j$, and the natural-language statement and unlocker action of the targeted graph node.

Curator prompt (forward phase, Sec. 4.2)

Role. You are guiding a blinded coding agent. The agent's next ReAct step is currently `candidate`, but you believe a small redirection would unlock more useful evidence. You may rewrite *at most one* step.
Visible to you. (a) The issue description d_i. (b) The full prefix h_{t+j} including all prior assistant messages, tool calls, and observations. (c) The target node v: a single statement and its unlocker action (e.g., "read `responses/security_groups.py` lines 183–197 to compare with sibling `describe_security_groups`"). (d) The candidate step `candidate` the blinded agent originally produced.
Hidden from you. The reference patch, the test suite, the rest of the graph, and any future steps. Do not speculate about them.
Task. Produce $y'_{t+j} = (c'_{t+j},\ a'_{t+j})$, a single ReAct response whose action moves the trajectory toward v's unlocker.
Hard rules. (C1) The reasoning c'_{t+j} must be entailed by what is visible in h_{t+j} — no claim that requires evidence the prefix has not yet produced. Frame the redirection as a natural next thought ("before going deeper into X, let me first check Y"). (C2) Every entity (file path, identifier, line number) you mention must already appear somewhere in h_{t+j}, either via the issue text or via earlier observations. If v's unlocker mentions an entity not yet observed, choose a smaller approach action that surfaces that entity first. (C3) The action a'_{t+j} is a single tool call from the agent's tool ontology (`str_replace_editor view`, `execute_bash`, `think`, `create_file`, `str_replace`); arguments must be fully specified. (C4) Do not write or refer to any meta-concept ("oracle", "graph", "target node", "reference patch"). Your output must read as if a skilled developer wrote it from scratch. (C5) Stay within one step. Do not stack multiple tool calls; do not pre-announce future steps in detail.
Output. JSON: $\{$response_content : c'_{t+j}, response_tool_calls : $[a'_{t+j}]\}$. Nothing else.

Claim-grounding judge prompt. The judge implements $\mathrm{Claim}(y'_{t+j}, h_{t+j}) \in \{0,1\}$ and is invoked only on the curator-mutated step. It runs after the symbolic check and is by construction blinded to $p^\star$ and to the target node.

Claim-grounding judge prompt (forward phase, Sec. 4.2)

Role. You are an independent reviewer. You decide whether a single proposed assistant step is *grounded* in the prior interaction history — meaning every claim it makes and every entity it references is either stated in the issue, visible in earlier tool outputs, or trivially derivable from them.
Inputs. (i) Issue d_i. (ii) Full untruncated prefix h_{t+j} (every assistant message and every observation). (iii) Proposed step y'_{t+j} (its reasoning text and its tool call).
Hidden. You do not have access to the reference patch, the test suite, or any oracle. Judge solely from (i)–(iii).
Decision criteria (all must hold). (J1) REASONING ENTAILMENT. Every assertion in the reasoning text is either present in d_i / h_{t+j} verbatim, or is an immediate one-step inference from content that is. Disallowed: identifying a root cause whose symptom has not yet been observed; asserting that a method "probably" does X without having read its body. (J2) ARGUMENT REACHABILITY. Every argument of the tool call (file paths, line numbers, symbol names, shell flags) appears in h_{t+j} or d_i, or is a transparent transformation thereof (e.g. widening a previously viewed line range by a few lines). (J3) NO ORACLE ARTIFACTS. The text contains no reference to "oracle", "golden patch", "reference solution", "ground truth", or any meta-concept about being guided. (J4) CONTINUITY. The step is a natural incremental continuation of h_{t+j}; there is no sudden jump in knowledge or unexplained change of focus.
Output. JSON: $\{$valid : true | false, reasons : $[\ldots]\}$, where `reasons` cites the violated criterion and the offending span when `valid`=false. Return `valid`=true iff (J1)–(J4) all hold.

Establishment verifier prompt. For each candidate window the verifier is invoked on every (v, h_τ) pair where v is a node not yet in $U_{\tau-1}$ and h_τ extends $h_{\tau-1}$ by exactly one step. It returns $\mathrm{Est}(v, h_\tau) \in \{0, 1\}$, feeding directly into U_τ, $\mathcal{A}_{\tau-1}$, and Prog_τ (Sec. 4.1).

Establishment verifier prompt (Sec. 4.1)

Role. Decide whether the trajectory prefix h_τ has *established* the graph node $v = (s_v, \eta_v, u_v)$, where s_v is a natural-language statement and u_v is a required interaction (e.g., a file view, a grep, a script execution).
Two-part criterion. Establishment requires both: (E1) ACTION MATCH. Some action in h_τ matches the requirement specified by u_v. Equivalence is judged by intent, not by string match: a `view` of a wider line range that includes the target lines counts; an `execute_bash python -c '...'` that exercises the same code path as a probe script counts; a `grep` over a superset of the requested directory counts. (E2) STATEMENT ENTAILMENT. The observation produced by the matching action, conditioned only on h_τ, entails s_v. You may NOT consult the repository directly — restrict the judgment to what the trajectory has actually surfaced. If the observation is consistent with s_v but does not entail it (e.g., a function name appears but its body is not shown), return false.
Inputs. d_i; the full prefix h_τ; the node v. You do NOT see $p^\star$, the test suite, or other graph nodes.
Output. JSON: {established : true | false, matched_action : (step index or null), evidence : (short quote from the matching observation), reason : (one sentence; required when false)}.

Termination, retries, and trajectory acceptance. A candidate that fails either half of Ground_t is dropped from $\mathcal{S}_t$ without re-prompting — the curator may not retry on the same (j, v) pair within a window, since unbounded retries would let the curator search for a leakage-free phrasing of an inherently leaky claim. If all mutated candidates fail and no pure seed clears η_t, the fallback rule of Sec. 4.2 commits the maximum-effectiveness pure seed. The forward loop terminates when the agent emits the `finish` action; the resulting patch is then run against $\mathcal{T}_i$ and the trajectory is admitted into $\mathcal{D}_{\text{ours}}$ iff all targeted tests pass.

I End-to-end worked example

This appendix traces a single SWE-Gym instance, `getmoto/moto#6041` (*"ec2.describe_security_group_rules does not use filter"*), end-to-end through the P2T pipeline. The instance is representative because (i) the oracle patch is one line, giving a clean contrast between a wandering blinded rollout and the curated trajectory; (ii) the blinded teacher over-engineers the fix on the model layer and breaks an existing test, so curation has to both *remove* a long detour and *insert* a small correction; and (iii) the distilled graph $G^\star$ contains every node category (static fact, dynamic fact, reproduction, analysis, plan, edit, validation), so a single instance illustrates the full recipe.

I.1 Issue x and oracle patch $p^\star$

Issue x

`ec2.describe_security_group_rules` does not use filter. Calling it with `Filters=[{"Name":"group-id","Values":[sg_id]}]` returns rules for *all* security groups (default + dummy) instead of only the dummy SG. The user's repro uses `boto3 + moto.mock_ec2` and expects exactly the default egress rule for the new SG; actually gets two.

Oracle patch $p^\star$ (one line)

```
-- a/moto/ec2/responses/security_groups.py
+++ b/moto/ec2/responses/security_groups.py
@@ class SecurityGroups(EC2BaseResponse):
  def describe_security_group_rules(self) -> str:
    group_id = self._get_param("GroupId")
-   filters = self._get_param("Filter")
+   filters = self._filters_from_querystring()
    rules = self.ec2_backend.describe_security_group_rules(group_id, filters)
```

The fix replaces a flat `_get_param("Filter")` (which looks for a literal `"Filter"` key in the querystring, finds none, and returns `None`) with the EC2 helper used by every other `describe_*` handler in the same file.

I.2 Phase 1 — Process graph distillation

The reverse phase converts $p^\star$ into a latent process graph $G^\star = (V, E)$ with **10 contextual-fact nodes** and **6 solution-milestone nodes** (1 reproduction, 1 analysis, 1 plan, 1 edit, 2 validations). The converged fact set $V_F^\star$ is summarised in Table 2 and the full DAG is shown in Fig. 5.

Table 2: Converged fact set $V_F^\star$ for `moto#6041`. Each node carries a natural-language statement, a type tag, and an explicit unlocker (omitted from this table for brevity; see Fig. 5).

id	type	statement
f_1	static	Repro shows `describe_security_group_rules` returning rules for *all* SGs when a `group-id` filter is passed.
f_2	static	Line 197 of the response handler uses `self._get_param('Filter')` to parse filters.
f_3	static	Sibling `describe_security_groups` (line 186) uses `self._filters_from_querystring()` – the standard EC2 pattern.
f_4	static	`_filters_from_querystring` in `EC2BaseResponse` parses the numbered `Filter.N.*` querystring into a `{name: values}` dict.
f_5	static	`_get_param(p)` does an *exact* key lookup; for `p='Filter'` it never matches `Filter.1.Name` and returns `None`.
f_6	dynamic	At runtime the querystring contains `Filter.1.Name/Filter.1.Value.1`; `_get_param('Filter')`→`None`, helper → correct dict.
f_8	static	Backend `describe_security_group_rules` delegates to `describe_security_groups(group_ids, filters)`.
f_9	static	Backend filter step (line 540: `if filters:`) is *skipped* when `filters` is `None`, so all groups match.
f_{10}	static	`_filters_from_querystring` is the standard pattern across *all* other EC2 `describe_*` handlers (`instances.py`, `hosts.py`, …).
f_{11}	static	The existing test masks the bug – it iterates *all* returned rules looking for an id match, never asserts the count.

One iteration of the proposer–critic loop. We illustrate non-leakage enforcement on a single round. Starting from $V^{(0)} = \{f_1\}$, the Proposer reads $p^\star$ and proposes $\Delta V_F^{(1)} = \{c_1, \ldots, c_5\}$ (Table 3).

Table 3: Proposer round 1 candidates for `moto#6041`.

cand.	proposed statement	critic verdict
c_1	Buggy line 197 uses `_get_param('Filter')`.	keep ($\rightarrow f_2$)
c_2	*The fix is to call `_filters_from_querystring()` at line 197.*	**prune (leaks $p^\star$)**
c_3	Sibling `describe_security_groups` uses the correct helper.	keep ($\rightarrow f_3$)
c_4	At runtime the querystring uses `Filter.N.*` keys, not `Filter`.	keep ($\rightarrow f_6$)
c_5	Backend silently skips filtering when `filters is None`.	keep ($\rightarrow f_9$)

The critic's feedback to the Proposer is

> *ϕ: "c_2 is the patch itself; its unlocker presupposes knowledge of $p^\star$. Decompose into (a) a fact identifying the bug location, (b) a fact identifying the correct alternative pattern observed elsewhere in the repo, and (c) a plan node that conjoins them."*

A second iteration adds $\Delta V_F^{(2)} = \{f_4, f_5, f_8, f_{10}, f_{11}\}$ to close residual gaps (the helper definition, the `_get_param` semantics, the backend delegation, the cross-file prevalence of the pattern, and the masking test); the loop converges. A subsequent ScaffoldArtifactDAG pass appends $V_A^\star = \{\text{repro}_1, \text{analysis}, \text{plan}, \text{edit}_1, \text{val}_1, \text{val}_2\}$ and links every node into the DAG of Fig. 5.

I.3 Phase 2 — Receding-horizon trajectory realization

We trace one sliding window in detail. Setup: window length $n = 10$, commit horizon $\delta = 4$, $K = 2$ blinded seeds plus up to one graph-aware mutation each (≤ 4 candidates per window), per-window floor η_t calibrated so that $\sum_t \eta_t \approx |V|$.

State at window start. The window we trace begins at prefix end-step $t = 22$ of the original blinded rollout. The realized-node set and available frontier are

$$U_{22} = \{f_1, \text{repro}_1, f_2\}, \qquad \mathcal{A}_{22} = \{f_3, f_5, f_8, f_{11}\}$$

(f_3, f_5, f_8 have $f_2 \in U_{22}$ as their only predecessor; f_{11} has $f_1 \in U_{22}$). This window is exactly where the blinded teacher takes its wrong turn: it omits f_3 (the sibling-method comparison) and is therefore biased to "fix" the backend rather than the response handler.

Two blinded seeds. Both seeds are length-10 continuations from $\pi_{\text{blind}}(\cdot \mid h_{22})$.

As a worked example of the Eff computation, take $s^{(0,f_3)}$. The initial frontier is $\mathcal{A}_{22} = \{f_3, f_5, f_8, f_{11}\}$ ($|\mathcal{A}| = 4$). The *mutation step* itself realizes only f_3 (Prog $= 1/4 = 0.25$).

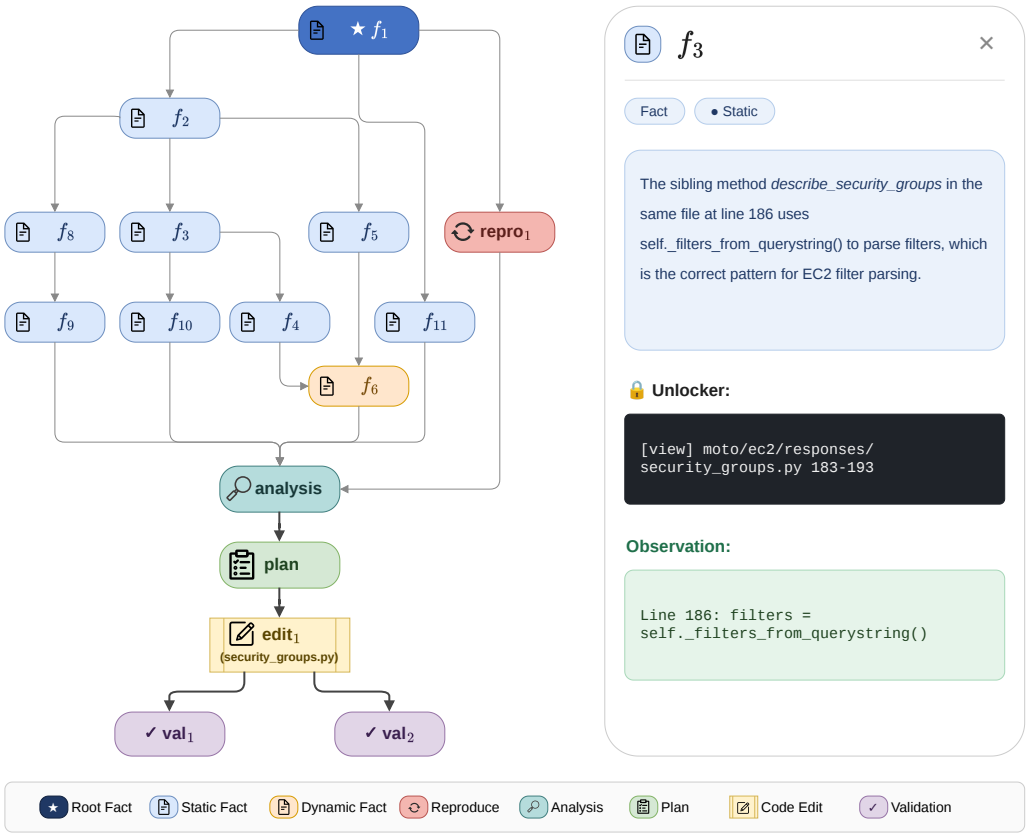


Figure 5: Distilled prerequisite graph $G^\star$ for `moto#6041`. Static facts (blue) form the contextual layer; the dynamic fact f_6 (orange) requires execution; the artifact layer (green/grey/red/purple) encodes the reproduction, analysis, plan, edit, and validation milestones. Edges denote prerequisite relations enforced by the critic during distillation.

Table 4: Window-22 candidate pool (n=10 steps each). $\mathrm{Eff}_{22} = \sum_\tau \mathrm{Prog}_\tau$ aggregates per-step ratios $\mathrm{Prog}_\tau = |\Delta_\tau|/|\mathcal{A}_{\tau-1}| \in [0,1]$; Len_{22} is response-token mass. Both mutated variants tie at $\mathrm{Eff}_{22} = 0.70$ above the floor $\eta_{22} \approx 0.5$; $s^{(0,f_3)}$ wins the length tie-break and is committed as $s^\star_{22}$.

candidate	Δ realized	$\lvert\Delta\rvert$	Ground	Eff_{22}	Len_{22}
$\tilde{s}^{(0,0)}$ (seed, model-first)	$\{f_8\}$	1	1 (no edit)	0.25	4.7k
$\tilde{s}^{(1,0)}$ (seed, test-first)	$\{f_{11}, f_5\}$	2	1 (no edit)	0.58	8.7k
$s^{(1,f_3)}$ (curation on seed 1)	$\{f_3, f_8, f_9\}$	3	1	0.70	5.4k
$s^{(0,f_3)} = s^\star_{22}$ (curation on seed 0)	$\{f_3, f_4, f_{10}\}$	3	1	**0.70**	**4.3k**

In the 7-step re-rolled suffix the blinded model – now seeing that the sibling handler uses `_filters_from_querystring` – naturally opens `_base_response.py` to learn what the helper does, realizing f_4 (frontier grew to $|\mathcal{A}| = 5$ after f_3, so $1/5 = 0.20$); a follow-up `grep -rn` for the pattern across `moto/ec2/responses/` realizes f_{10} ($1/4 = 0.25$). Summing the three per-step ratios gives $\mathrm{Eff}_{22} \approx 0.70$. The leakage rule rejects f_9 in $\tilde{s}^{(0,0)}$: even though step 26's view of the model file would expose it in the same observation as f_8, f_9's prerequisite (f_8) is not yet in $U_{\tau-1}$, so it contributes 0 and can be realized only at a later window.

Single-curation construction. The curator scans positions $j = 0, \ldots, 9$ along seed $k = 0$ and picks $j = 2$ (the first non-trivial branching point: the seed wastes step 25 on an empty assistant thought). Among $\mathcal{A}_{24} = \mathcal{A}_{22} = \{f_3, f_5, f_8, f_{11}\}$ (steps 23–24 of seed $k = 0$ realize no new node; f_8 would be realized only at step 26 of the unmutated seed via the model-file view, so at $j = 2$ it is still on the

frontier), the $v = f_3$ has is chosen as the curation target node. The replacement response and its critic verdicts are:

Mutated step y'_{22+2} (curator-authored, target f_3)

Assistant: "Before diving deeper into the model, let me first check how the nearby `describe_security_groups` method in the same response file parses its filters — it's the closest sibling and should reveal the standard pattern."

Tool call: `execute_bash cmd="grep -n 'def describe_security_groups\|filters' moto/ec2/responses/security_groups.py | head -20"`

Symbolic referential-integrity check. Entities mentioned in y' – `describe_security_groups`, `filters`, `moto/ec2/responses/security_groups.py`, the `grep` flag `-n` – all appear in $\mathrm{Obs}(h_{24})$ (the issue text and the views at steps 22 and 24). ✓ **PASS.**

Neural claim-grounding judge. The claim "the sibling method is in the same file and is a similar operation" is entailed by the view at step 22 (lines 190–210), where the tail of `describe_security_groups` is visible just above the buggy method. ✓ **PASS.** Therefore $\mathrm{Ground}_{22}(s) = 1$. A counterfactual mutation targeting f_5 with the claim "`_get_param` does exact-key matching, which is why `Filter` fails" would ✗ **FAIL** the neural judge – that semantics has not yet been observed – and would zero out Eff.

Selection and commit. The two mutated variants both clear three new nodes and tie at $\mathrm{Eff}_{22} = 0.70$, well above the floor $\eta_{22} \approx 0.5$, but they realize *different* parts of the graph: $s^{(0,f_3)}$ extends along the response-handler chain ($\{f_3, f_4, f_{10}\}$, branching from f_3 into the helper definition and its cross-file prevalence), whereas $s^{(1,f_3)}$ extends along the model-layer chain ($\{f_3, f_8, f_9\}$, since the seed-1 prefix had already oriented the agent toward the backend). Both pure seeds fall below the floor. The two mutated variants thus form the local non-dominated set on the Eff axis, and the tie-break by length selects the seed-0 mutation, which has the shorter prefix ($\mathrm{Len} = 4.3$k vs 5.4k tokens). $s^{(0,f_3)}$ is committed as $s^\star_{22}$ (Fig. 6). The first $\delta = 4$ steps are appended to the trajectory; the remaining suffix is replanned from the new prefix. The non-selected candidates are rolled back and never enter the trajectory.

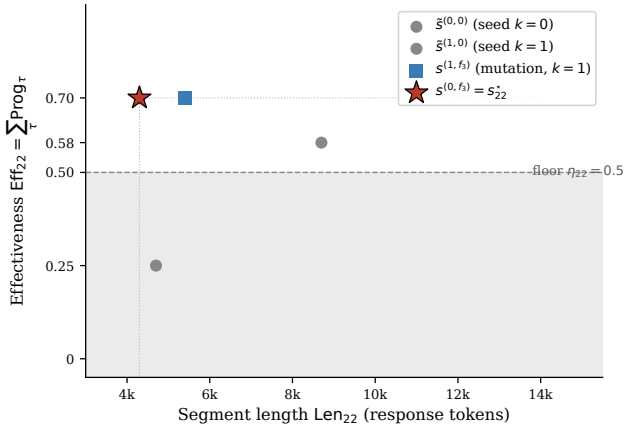


Figure 6: Window-22 candidate pool in $(\mathrm{Len}_{22}, \mathrm{Eff}_{22})$ space, with Len_{22} measured in response tokens (assistant messages plus their observations). Both pure seeds (grey circles) sit far below the floor $\eta_{22} = 0.50$ (shaded band, dashed line). The two mutated variants tie at $\mathrm{Eff} = 0.70$; the tie-break by length picks the seed-0 mutation (red star), which is committed as $s^\star_{22}$.

Side-by-side comparison. Fig. 7 contrasts the unmodified blinded rollout (steps 23–32) with the P2T trajectory (steps 23–37) over the same window. The committed prefix is the first $\delta = 4$ steps of $s^\star_{22}$ – step 23, step 24, the mutated y' at step 25, and the re-rolled step 26 that immediately benefits from y' – and only the assistant message at step 25 is curator-authored; every other committed token is a blinded continuation. Follow-on windows then replan from the new prefix and converge on

the one-line response-handler fix, whereas the unmodified rollout commits to a 100-step backend over-engineering loop that edits `SecurityRule`, the response template, and breaks an existing test.

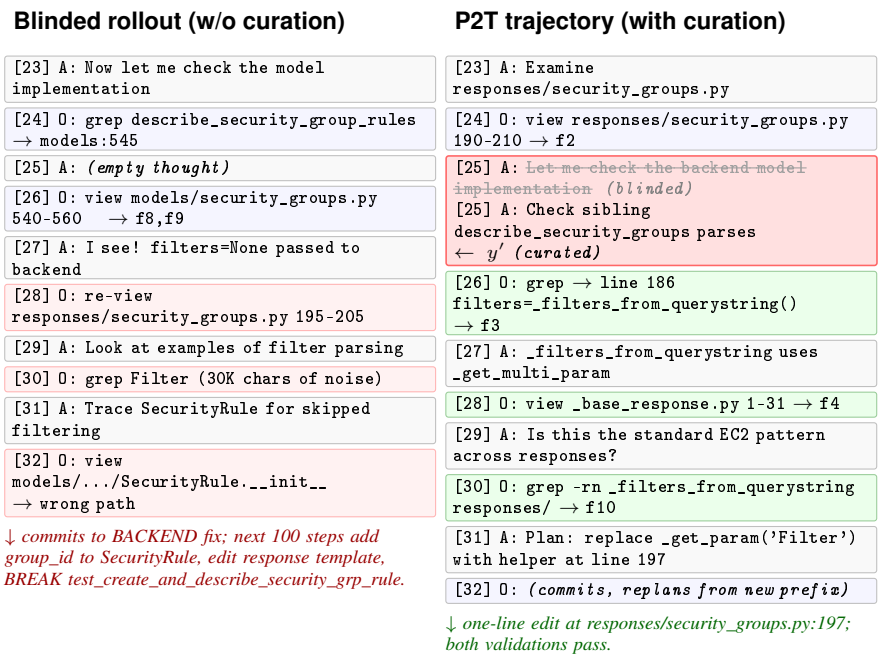


Figure 7: Side-by-side fragment of the same window in π_{blind} (left) and P2T (right). Rows labelled `A` are assistant responses; rows labelled `O` are environment observations. The single curator-authored step (red, y' at $j{=}2$) replaces an empty assistant thought with a frontier-aware inspection of the sibling `describe_security_groups` method, unlocking f_3. The 7-step re-rolled blinded suffix then organically realizes only the two facts naturally implied by that observation (f_4, f_{10}, green steps), whereas the unmodified rollout (red steps) wanders into the model layer and breaks an existing test.

J Training details

Student SFT. We fine-tune Qwen2.5-Coder-14B/32B-Instruct [3] with **ms-swift**'s Megatron-LM backend [32]. Full-parameter SFT runs for 3 epochs with Adam (lr 2×10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay 0.01, gradient clipping 1.0), global batch size 4 with sequence packing up to $131{,}072$ tokens, linear warm-up over the first 5% of steps and cosine decay to 0, and BF16 mixed precision. The loss is computed on assistant tokens only.

Context window and parallelism. Curated trajectories frequently exceed the base $32{,}768$-token window of Qwen2.5-Coder; we extend the effective context to $131{,}072$ tokens via YaRN positional interpolation [12] (factor 4) for both training and inference. Training uses tensor parallelism 2, context parallelism 4, and Megatron-style sequence parallelism.

Compute resources. All experiments—curation, SFT training, and SWE-bench evaluation—are run on a single node with $8\times$ NVIDIA H200 (141 GB) GPUs. Teacher rollouts and student inference are served via vLLM on the same node, and the curation pipeline (reverse-phase graph distillation and forward-phase trajectory construction) is executed on the same hardware.

K Compute-matched rejection-sampling baseline

The size-matched control in Sec. 5.2 fixes the number of curated trajectories but ignores the GPU-hours P2T spends on graph distillation, blinded seeding, mutation, and gating. A skeptical reader may ask whether the same compute, redirected into additional plain rollouts, would close the gap. This appendix runs that experiment under a strict compute-parity protocol on the same hardware

(single $8\times$H200 node, vLLM-served Qwen3-Coder-480B teacher, OpenHands scaffold, 100-iteration ReAct budget).

Compute accounting. End-to-end P2T curation over the 1.8k SWE-Gym instances costs ≈ 226 GPU-hours (reverse-phase proposer/critic + forward-phase blinded seeds, mutations, rollbacks, and per-step LLM gating). One full pass of plain teacher rollouts over the same SWE-Gym pool costs ≈ 68 GPU-hours. We therefore allocate the rejection-sampling baseline a budget of 4 rollout passes (≈ 272 GPU-hours, $\sim 20\%$ *more* than P2T), aggregate the resolved trajectories across passes (deduplicating per instance by keeping the shortest passing trajectory), and SFT Qwen2.5-Coder-32B-Instruct on the result under the same recipe as Sec. 5.1. As an upper bound on the supervision the 4-run baseline could in principle harvest, the cumulative resolve rate (Pass@4) over the 4 rollout passes on the SWE-Gym training pool reaches 34.3%, vs. a per-run Pass@1 of $\sim 29\%$; the additional passes thus recover only a sub-set of new instances rather than uniformly improving every trajectory.

Phase-level cost breakdown. Table 5 decomposes P2T's curation budget by phase. Process-graph distillation (Phase 1) accounts for 14.2% of the total and trajectory realization (Phase 2) for 85.8%. End-to-end, P2T costs $3.32\times$ one ordinary rejection-sampling pass, and Phase 2 alone $2.85\times$. This is a one-time data-curation cost; as shown below, it is more compute-effective than spending a comparable or larger budget on additional rejection-sampling passes.

Table 5: **Curation cost by phase** (aggregate H200 GPU-hours on a single $8\times$H200 node, SWE-Gym curation pool, Qwen3-C-480B teacher). Phase 1 is process-graph distillation; Phase 2 is trajectory realization for P2T and rollout collection for rejection sampling.

Method	**Phase 1**	**Phase 2**	**Total**
Test-Pass RS (1 pass)	–	68	68
Test-Pass RS (4 passes, compute-matched)	–	272	272
P2T	32	194	226

Results. Table 6 compares the compute-matched RS baseline against P2T (full) on SWE-bench Verified, reporting Pass@1, per-instance inference cost, and total curation GPU-hours.

Table 6: **Compute-matched comparison on SWE-bench Verified** (Qwen3-C-480B teacher, Qwen2.5-C-32B student). The RS baseline is allocated 4 rollout passes (≈ 272 GPU-hours, $\sim 20\%$ more than P2T's curation budget); resolved trajectories are aggregated across passes (shortest passing kept per instance) and used as SFT data under the same recipe as Sec. 5.1. **Bold** denotes the better cell.

Recipe	**Pass@1 (%)** $\uparrow$	**Cost ($)** $\downarrow$	**Curation GPU-hours** $\downarrow$
Test-Pass RS ($4\times$ rollouts)	43.2	0.86	≈ 272
P2T (full)	**50.4**	**0.78**	$\approx$ **226**

Even with $\sim 20\%$ more curation compute, the rejection-sampling baseline remains bounded by its own ceiling: the additional rollout passes recover supervision only on instances the teacher happens to solve at least once across 4 tries, and contribute nothing to the per-step quality of the trajectories that do get retained. P2T spends comparable GPU-hours but allocates them differently—toward distilling $G^\star$ and shaping each retained trajectory along $(\mathrm{Eff}, \mathrm{Len})$—which is what unlocks the simultaneous Pass@1 gain and per-instance inference-cost reduction in Table 1.

L Multi-seed robustness of the main results

Table 1 reports a single training run per configuration. To quantify how much of the reported gain could be attributed to training randomness and evaluation variance, we repeat student SFT with three independent random seeds for the Qwen3-C-480B curator.

Protocol. Regenerating a complete P2T trajectory dataset costs ≈ 226 H200 GPU-hours (App. K), so we hold each curated dataset fixed and vary only the SFT seed. For each of the three recipes that isolate the contribution of P2T—Test-Pass RS, P2T (size-matched), and P2T (full)—and each student (Qwen2.5-C-14B/32B), we run full-parameter SFT three times with independent random seeds under the recipe of App. J, keeping the dataset and all hyperparameters fixed. Every resulting model is re-evaluated on SWE-bench Verified and Lite with one greedy rollout per instance under the protocol of Sec. 5.1. We report the mean $\pm$ sample standard deviation over the three independently trained models. The SWE-Lego baseline and the GLM-5-FP8 curator, which Table 1 reports from a single run, are not covered by this study.

Table 7: **Three-seed SFT results** (Qwen3-C-480B curator; mean $\pm$ sample std over three independently trained students, one greedy rollout per instance). Curated datasets are held fixed; only the SFT seed varies. **(+X.X)** reports the mean Pass@1 gain over Test-Pass RS.

Student	Training data	SWE-bench Verified		SWE-bench Lite	
		Pass@1 (%) ↑	Cost ($) ↓	Pass@1 (%) ↑	Cost ($) ↓
Qwen2.5-C-32B	Test-Pass RS	39.27 ± 0.76	0.923 ± 0.035	28.67 ± 0.35	0.933 ± 0.006
	P2T (size-matched)	41.87 ± 0.61 (+2.60)	0.863 ± 0.023	29.57 ± 0.23 (+0.90)	0.890 ± 0.010
	P2T (full)	$\mathbf{49.67 \pm 0.70}$ (+10.40)	$\mathbf{0.793 \pm 0.012}$	$\mathbf{35.80 \pm 0.17}$ (+7.13)	$\mathbf{0.813 \pm 0.015}$
Qwen2.5-C-14B	Test-Pass RS	35.87 ± 0.42	0.923 ± 0.025	22.57 ± 0.51	0.917 ± 0.021
	P2T (size-matched)	37.20 ± 0.40 (+1.33)	0.873 ± 0.035	23.23 ± 0.50 (+0.66)	0.917 ± 0.015
	P2T (full)	$\mathbf{41.67 \pm 1.50}$ (+5.80)	$\mathbf{0.793 \pm 0.042}$	$\mathbf{29.77 \pm 0.40}$ (+7.20)	$\mathbf{0.833 \pm 0.015}$

Results. Table 7 confirms that the gains in Table 1 are not an artifact of a favorable training run. Relative to Test-Pass RS, P2T (full) improves mean Pass@1 by $+10.40$/$+7.13$ points for the 32B student and $+5.80$/$+7.20$ points for the 14B student on Verified/Lite, margins several times larger than the seed-to-seed standard deviation (≤ 1.50 points), while reducing mean inference cost in every setting. P2T (size-matched) also remains ahead of Test-Pass RS in all four Pass@1 comparisons, so the per-trajectory quality effect isolated in Sec. 5.2 persists across seeds. The single-run values of Table 1 lie within the spread of the corresponding three-seed estimates.

M Evaluation on SWE-bench Pro

SWE-bench Verified and Lite are dominated by single-file edits and may be affected by pre-training contamination. To test whether the benefit of P2T carries over to harder, longer-horizon tasks, we additionally evaluate on SWE-bench Pro [2]. Because our students are trained only on Python SWE-Gym instances, we restrict evaluation to the 266-task Python subset of the public SWE-bench Pro set.

Protocol. We compare the Qwen2.5-C-32B students trained on Test-Pass RS and on P2T (full) data from Sec. 5.2 (Qwen3-C-480B curator), evaluated with one greedy rollout per task under the same OpenHands scaffold. The official SWE-bench Pro evaluation scripts suggest a 150-turn budget; we keep the 100-turn budget used throughout this paper to match the SFT setting. Cost is metered as in Sec. 5.1. Before SFT, the base Qwen2.5-C-32B resolves none of the 266 tasks.

Table 8: **SWE-bench Pro (Python subset, 266 tasks)** for the Qwen2.5-C-32B student under the 100-turn OpenHands budget. The base model resolves $0/266$ tasks before SFT. **Bold** denotes the better cell.

Training data	Pass@1 (%) ↑	Cost ($) ↓	Avg. turns ↓	Hit 100-turn cap ↓
Test-Pass RS	8.27	0.982	94.53	192/266 (72.2%)
P2T (full)	**10.52**	**0.965**	**90.88**	**178/266 (66.9%)**

Results. On this substantially harder benchmark, P2T improves Pass@1 by $+2.25$ points, reduces average turns by 3.9%, and lowers the incidence of exhausting the turn budget by 5.3 points (Table 8).

The cost reduction is modest (1.7%): because both students frequently exhaust the 100-turn budget, per-instance cost is heavily truncated, which compresses the observable cost difference. The gains are smaller in absolute terms than on Verified and Lite but point in the same direction on both axes. Evaluation on multilingual and multi-repository benchmarks such as SWE-bench Multilingual and Multi-SWE-bench [27] remains open (App. P).

N Pipeline yield by stage

Table 9 traces the SWE-Gym pool through each stage of P2T. Of the 2,438 instances, 312 lack an executable Docker environment and a further 248 fail reference-patch verification, $\mathcal{T}_i(p_i^\star) = 1$. Phase 1 then removes 63 instances whose raw process graph exceeds the admission limit (at most six proposer–critic rounds and 27 nodes) before forward realization. Forward construction produces a terminal trajectory for every remaining instance, using the candidate fallback when needed, and 109 of these trajectories (6.0%) fail final patch verification and are discarded. The "1.8k" training pool in Sec. 5.1 therefore refers to the 1,815 curation-eligible instances with an admitted process graph, and the "1.7k" curated trajectories to the 1,706 that pass the final admission gate and form P2T (full). Rejection-sampling baselines can use all 2,126 executable instances because they do not require a valid reference patch.

Table 9: **Per-stage yield of the P2T pipeline** on SWE-Gym (Qwen3-C-480B teacher).

Stage	Retained / input	Retention	Criterion
SWE-Gym environment check	2,126 / 2,438	87.2%	Executable Docker environment
Reference-patch verification	1,878 / 2,126	88.3%	$\mathcal{T}_i(p_i^\star) = 1$
Process-graph construction	1,815 / 1,878	96.6%	≤ 6 rounds and ≤ 27 nodes
Forward construction	1,815 / 1,815	100%	Terminal trajectory (with fallback)
Final patch verification	1,706 / 1,815	94.0%	Passing trajectory admitted

O Human audit of residual leakage

The groundedness checks in both phases are LLM-mediated, so their non-leakage guarantee is empirical. To measure residual leakage directly, we conducted a blinded audit with two independent expert programmers, each with more than five years of Python experience and blinded to the system verdicts.

Protocol. *Process-graph nodes.* We sampled five nodes from each of five instances (25 nodes in total). Experts received the issue, the relevant pre-fix code, the reference patch for comparison, and each node together with its predecessor states and unlocker. They judged whether the node statement or its unlocker relied on patch-only information rather than evidence discoverable from the issue, the code, and the predecessor nodes. *Accepted curator mutations.* We sampled five curator-mutated responses that passed the Phase 2 groundedness check from each of six trajectories (30 in total). To let experts inspect the complete preceding history, we restricted sampling to mutations within the first 30 interactions. Experts judged whether each mutation introduced information unsupported by that history.

Table 10: **Blinded expert audit of residual leakage.** The leakage proportion is the mean over the two experts; agreement is Cohen's κ.

Audited artifact	Items	Mean leakage proportion	Cohen's κ
Process-graph nodes	25	0.060	0.648
Accepted curator-mutated responses	30	0.083	0.783

Results. Table 10 shows low but nonzero residual leakage at both stages, with substantial inter-annotator agreement. The second figure measures end-to-end leakage after the hybrid neural–symbolic

check, not the error rate of the neural judge in isolation. We audited only accepted mutations, which carry the higher-risk error: a falsely accepted mutation can place unsupported content in a retained trajectory, whereas a falsely rejected one only removes a candidate while the original blinded seeds remain in the pool. The false-rejection rate is therefore not measured. Given the small sample, these results bound residual leakage empirically rather than provide a formal guarantee.

P Limitations and outlook

Limitations. P2T inherits five constraints worth flagging. (i) *Privileged-signal availability.* The reverse phase requires a reference patch $p^\star$ and an executable test suite per instance. Both hold on SWE-Gym and SWE-bench but exclude issues without a maintained CI; extending the recipe to such issues would require a surrogate verifier (e.g., teacher-generated patches gated by self-consistency or property-based tests). Patch reliability is an input requirement rather than something P2T establishes: synthetic tasks with an execution-validated golden patch (e.g., the inverse-injection patches of SWE-smith or R2E-Gym) satisfy it, but P2T does not validate, correct, or derive supervision from imperfect or unverified model-generated patches, and such weakly labeled settings are outside its current scope. (ii) *LLM-mediated guarantees.* Both the non-leakage critic in Phase 1 and the groundedness/establishment judges in Phase 2 are LLM-instantiated. Our value-of-information and component ablations (App. C, Sec. 5.4) show the resulting graphs and gates carry the right signal in aggregate, but per-instance correctness is empirical rather than formal. Our blinded human audit (App. O) finds low but nonzero residual leakage (0.060 of sampled graph nodes and 0.083 of accepted curator mutations), and the critic's false-rejection rate, i.e., how often it rejects a valid grounded step, is not measured. A tighter, model-agnostic certificate of minimality and grounding remains open. (iii) *Curation compute.* Distillation and forward realization cost on the order of 10^2 GPU-hours for 1.8k instances (App. K); while compute-matched against rollout scaling, this is non-trivial and scales with graph size. (iv) *Scaffold and student scope.* Experiments fix OpenHands as the scaffold and the Qwen2.5-Coder family as the student; transfer to agentless pipelines, browser-augmented agents, or non-Qwen students is not yet validated. Likewise, our evaluation covers Python benchmarks only (SWE-bench Verified, Lite, and the Python subset of SWE-bench Pro, App. M); performance on multilingual issue resolution remains to be established. (v) *Reproducible environments.* Like SWE-bench and SWE-Gym, P2T assumes a buildable, fixed Docker image per instance. Candidate segments run from isolated snapshots and every committed patch is re-tested before admission, which isolates state across candidates but does not resolve intrinsically flaky tests, external network failures, races, or unseeded randomness. This assumption is shared by Test-Pass RS and SWE-Lego, so nondeterminism limits the scope of all compared methods rather than P2T alone.

Outlook. Two extensions follow naturally. First, $G^\star$ is a generic latent representation of process supervision: training a process reward model to score per-step compliance with $G^\star$ would lift the signal from behavior cloning to online RL while preserving non-leakage, addressing a known weakness of SFT under distribution shift. Second, the privileged-signal $\rightarrow$ latent-structure $\rightarrow$ blinded-realization factorization is not specific to code; any task with a verifiable terminal artifact and a partially observable, tool-mediated solution process (theorem proving, scientific protocol execution, multi-step data analysis) admits the same recipe. Together, these directions suggest that the right unit of supervision for capable agents is neither the trajectory nor the terminal artifact in isolation, but the *structure* that links them.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the contributions.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a discussion of its limitations in appendix.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The only formal result is the segment-greedy Pareto-optimality claim referenced in Sec. 4; its assumptions (additivity of Eff and Len, matching floor calibration $\sum_t \eta_t = \eta_i$, deterministic per-window floor satisfaction) and a proof sketch are stated in App. E, together with an explicit discussion of the gap to global optimality.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides detailed descriptions of the experimental setup, including the P2T curation framework, prompt, SWE-Gym training, and SWE-bench Verified/Lite evaluation across two teachers, allowing for reproducibility of the main results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Yes, we will open-source the code, and the SWE-Gym data.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper provides detailed descriptions of the experimental setup, including the P2T curation framework, prompt, SWE-Gym training, and SWE-bench Verified/Lite evaluation across two teachers, allowing for reproducibility of the main results. Detailed training args are in appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: For the Qwen3-C-480B curator, we repeat student SFT with three independent random seeds on fixed curated datasets and report mean $\pm$ sample standard deviation on SWE-bench Verified and Lite (App. L); the variability captured is training randomness, as re-curating the datasets is prohibitively expensive. Other cells of Table 1 are single runs. We also report paired Mann–Whitney tests for the trajectory-quality comparison (Sec. 5.3).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All experiments (data curation, SFT training, and SWE-bench evaluation) are run on a single node with $8\times$ NVIDIA H200 GPUs, as documented in App. J.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

Justification: The work is primarily focused on data curation and model training for SWE-bench evaluation, which does not have direct societal impacts.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release any models or datasets that pose a high risk for misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All existing assets used are cited at first use, accessed under their original open-source licenses (MIT/Apache-2.0), and used within their intended terms of use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were used only for grammar checking and light editing of the manuscript and do not impact the core methodology, scientific rigor, or originality of the research.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.