Quality, Not Just Outcome: A Structural Scoring Framework for Selecting Coding-Agent Trajectories for Supervised Fine-Tuning

Sida Li[1] Yuanxin Liu[1] Shicheng Li[1] Linli Yao[1] Xu Sun[1]
Haoli Bai[2] Xiao-Hui Li[2]

[1]National Key Laboratory for Multimedia Information Processing, Peking University
[2]Huawei Technologies Co., Ltd.

{lisida,liuyuanxin,linliyao}@stu.pku.edu.cn {lisc99,xusun}@pku.edu.cn
{baihaoli,lixiaohui33}@huawei.com

Abstract

Open coding agents are commonly trained through supervised fine-tuning on interaction trajectories whose final patches pass the benchmark test suite. However, this outcome-level filtering criterion ignores process quality: trajectories that eventually succeed may still contain inefficient interaction patterns such as repetitive file inspection, unstable tool usage, or delayed modification behavior, all of which become part of the imitation target. We propose TQS (Trajectory Quality Score), a structural scoring framework for coding-agent trajectories that operates directly on trajectory structure, requires no LLM judge, and runs in linear time. TQS characterizes trajectory quality through structural signals organized around structured convergence: information focus, intent–action alignment, and policy dynamics. Using 155,144 successful trajectories from a public open-license corpus, we rank trajectories by TQS and fine-tune coding agents on filtered subsets. Models trained on top-ranked trajectories consistently outperform those trained on random or bottom-ranked subsets on SWE-Bench Verified. Our results suggest that process-level trajectory quality is an important factor in coding-agent supervision and that structural trajectory filtering provides an effective and scalable alternative to LLM-based data selection. Project page: https://github.com/DuHeng0128/TQS.git.

1 Introduction

Coding agents that interact with sandboxed environments to inspect repositories, execute tests, and edit files have recently achieved strong performance on SWE-Bench Verified-style software engineering tasks (Jimenez et al., 2024; OpenAI, 2024). Beyond advances in foundation models and inference-time scaffolding, an increasingly important ingredient in these systems is trajectory supervision: training on multi-turn agent demonstrations consisting of reasoning traces, tool calls, tool observations, and final patches (Pan et al., 2025; Ma et al., 2024; Ariyak et al., 2026). For medium-scale open-weight models, a common and effective practice is to collect such trajectories from more capable teacher agents and retain those whose final patches pass the benchmark test suite.

However, outcome-level correctness does not necessarily imply process-level quality. Among successful trajectories, behavioral patterns vary substantially. Some trajectories localize the bug quickly, maintain coherent search scope, and transition efficiently from exploration to modification. Others repeatedly inspect the same files, interrupt stalled shell commands, or postpone meaningful edits until late in the interaction budget, yet still eventually reach a passing patch through extended trial-and-error. These differences matter because trajectory-level supervised fine-tuning exposes the student to the full interaction process, including reasoning traces, tool use, and the final patch (Kamoi et al., 2024). Inefficient or unstable interaction patterns therefore become part of the training signal.

The instruction-tuning literature has shown that curated subsets can outperform larger, noisier training corpora (Zhou et al., 2023; Chen et al., 2024; Liu et al., 2024; Li et al., 2024b; Xia et al., 2024b). Coding-agent trajectories pose a different selection problem. They are long interaction histories whose ordering records exploration, editing, testing, and recovery. Perplexity-style scores are confounded by length and tokenization, while LLM-based judges are costly at the scale of hundreds of thousands of trajectories and billions of tokens. Furthermore, these approaches are not designed to characterize how effectively an agent organizes its search process over time.

In this work, we propose TQS (Trajectory Quality Score), a structural scoring framework for OpenHands-style coding-agent trajectories. The framework is built around *structured convergence*:

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 48484–48500
October 24-29, 2026 ©2026 Association for Computational Linguistics

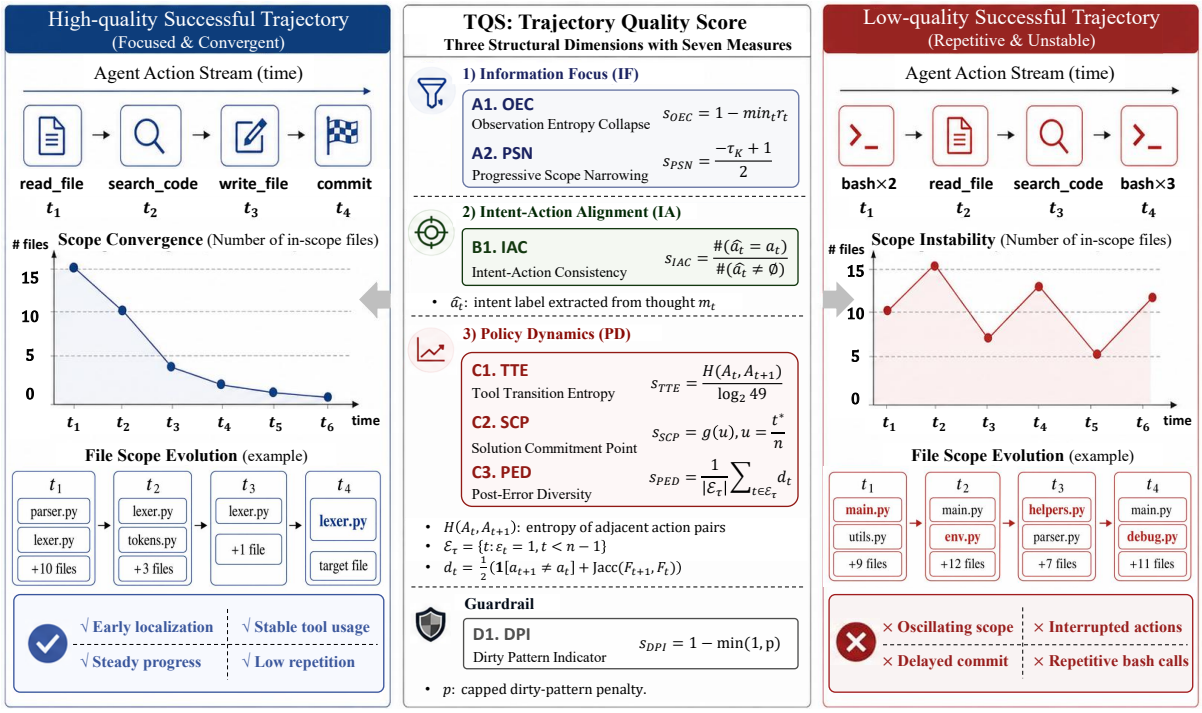


Figure 1: **Overview of TQS.** Contrasting high-quality (focused and convergent) with low-quality (repetitive and unstable) successful trajectories, TQS evaluates trajectory quality along three structural dimensions: Information Focus (OEC, PSN), Intent–Action Alignment (IAC), and Policy Dynamics (TTE, SCP, PED), with a Dirty Pattern Indicator (DPI) guardrail. These measures are aggregated into a weighted score in $[0, 1]$.

as an agent approaches a solution, a strong trajectory becomes increasingly organized around the relevant fix. This organization appears at three levels. The agent should focus its observations and file operations on task-relevant evidence, keep its stated intent aligned with its tool actions, and move from broad exploration toward productive edit, test, and recovery behavior. We operationalize these levels through seven trajectory-level measures, each computable in linear time from event logs alone. Outcome labels are used only to validate candidate structural signals during metric development; once the components are fixed, TQS ranks successful trajectories without inspecting the final patch, held-out test result, or downstream SFT performance.

Empirically, we score 155,144 successful trajectories from CoderForge-Preview (Ariyak et al., 2026) and construct matched top-ranked, bottom-ranked, and random subsets from the eligible fixed-budget SFT pool under task- and repository-level diversity constraints. We LoRA-fine-tune the same Qwen3-Coder-30B-A3B-Instruct model (Ariyak et al., 2026; Yang et al., 2025a; Hu et al., 2022) on each subset using identical hyperparameters. Models trained on top-ranked trajectories consistently outperform those trained on random or bottom-ranked subsets on SWE-Bench Verified (OpenAI, 2024) under the OpenHands framework (Wang et al., 2025). These results suggest that trajectory-level process structure is an important and previously underexplored dimension of coding-agent supervision.

Our contributions are threefold: (i) we formulate coding-agent trajectory quality through structured convergence and operationalize it with seven efficient trajectory-level measures; (ii) we introduce a scalable framework for trajectory scoring and filtering without relying on LLM judges or external reward models; and (iii) we present an end-to-end open pipeline for trajectory scoring, filtering, fine-tuning, and evaluation, demonstrating that quality-aware trajectory selection improves downstream coding-agent performance.

2 Related Work

Coding agents and trajectory supervision. Recent coding agents interact with repositories through tool-augmented environments that support file inspection, shell execution, testing, and patch generation. Frameworks such as SWE-agent (Yang et al., 2024) and OpenHands (Wang et al., 2025) standardized the agent-computer in-

teraction pipeline and enabled large-scale collection of executable agent trajectories. Building on these infrastructures, datasets such as SWE-Gym (Pan et al., 2025), Lingma SWE-GPT (Ma et al., 2024), and CoderForge-Preview (Ariyak et al., 2026) progressively scaled teacher-generated trajectory supervision for open-weight coding agents. Recent work has also explored trajectory synthesis and reward-guided curation for software and GUI agents (Sun et al., 2025). While reinforcement-learning approaches for software engineering agents (Wei et al., 2025) further improve exploration and decision-making, they typically still rely on strong supervised initialization, making trajectory quality important for both SFT and RL-based pipelines.

Data selection for supervised fine-tuning. Data-selection methods for supervised fine-tuning usually use an external judge or a training-derived importance signal. AlpaGasus (Chen et al., 2024) and DEITA (Liu et al., 2024) ask stronger LLMs to score samples. LESS (Xia et al., 2024b), Cherry-LLM (Li et al., 2024b), and Superfiltering (Li et al., 2024a) estimate utility through gradient influence or model-internal signals; related methods have also been applied to code instruction tuning (Wang et al., 2024). These selectors were developed for short, single-turn samples. Applying them to an agent trajectory would require redefining their sample-level signals over a long interaction history. TQS instead uses the temporal structure already present in the event log.

Process-level evaluation and trajectory quality. Recent work has increasingly emphasized evaluating intermediate reasoning and interaction processes rather than final outcomes alone. Prior studies on chain-of-thought faithfulness analyze whether intermediate reasoning traces are consistent with the eventual answer (Lanham et al., 2023; Bentham et al., 2024). AgentDiagnose (Ou et al., 2025) applies an LLM rubric to browser-agent trajectories and shows that process-aware trajectory selection can improve training efficiency. Beyond this, several methods use entropy or behavioral diversity to analyze and optimize agent policies (Chen et al., 2026; Li et al., 2026; Dong et al., 2025; Yu et al., 2025a). Recent work has also explored process supervision and trajectory curation for software-engineering agents using privileged patch-level information (Ma et al., 2026).

3 Trajectory Quality Score

3.1 Preliminaries and Notation

A trajectory $\tau = (s_0, s_1, \ldots, s_{n-1})$ is a sequence of n steps. Each step is represented as

$$s_t = (a_t, m_t, o_t, F_t, \varepsilon_t),$$

where $a_t \in \mathcal{A}$ is an action type (one of `think`, `read_file`, `write_file`, `execute_bash`, `search`, `submit`, `other`); m_t is the free-text thought that precedes the action; o_t is the tool observation returned after the action; $F_t \subseteq \mathcal{F}$ is the set of file paths touched by the action; and $\varepsilon_t \in \{0, 1\}$ indicates whether o_t matches a predefined error pattern. This tuple is recoverable from any OpenHands-style event log without model inference.

3.2 Design Intuition and Score Construction

A coding-agent trajectory is a search process over observations, files, actions, and intermediate hypotheses. We view a high-quality trajectory as one that exhibits *structured convergence*: over the course of the interaction, the search becomes organized around the emerging fix. TQS relies on event-log evidence that the agent is narrowing its search toward a task-relevant region. At minimum, it provides a low-cost way to filter out clearly undesirable trajectory patterns.

Structured convergence has three observable levels. **(A) Information focus** asks whether the agent narrows the relevant evidence through its observations and file operations. **(B) Intent–action alignment** asks whether the stated thought before a tool call points in the same direction as the action that follows. **(C) Policy dynamics** asks whether the interaction policy develops a productive rhythm of exploration, execution, and error recovery. Some failures are categorical rather than gradual; the trajectories associated with them are inefficient. A run may trigger an early-stop pattern, never make a substantive edit, or repeat the same failing command. DPI therefore sits outside the three-dimensional organization as a guardrail for these cases.

The axes motivate candidate measures, and the selection protocol decides which ones enter TQS. Each component is computable from the step tuple in $O(n)$ CPU time without auxiliary LLM judging, so the scorer can process event logs spanning roughly 6B tokens. The checks in §3.4 retain measures that discriminate above chance, have limited correlation with step count, and preserve behavioral diversity during subset selection.

3.3 Components and Aggregation

TQS uses seven components, each normalized to $[0, 1]$ with larger values indicating cleaner structure. A component is left undefined when its required event is absent; Eq. 1 handles these cases during aggregation.

(A1) Observation Entropy Collapse (OEC). For each observation o_t, we compute character-level Shannon entropy (Shannon, 1948). Let $\widetilde{H}_t$ be its three-step moving average, $\overline{H}_{\text{open}}$ the mean entropy of the first $\min(5, n)$ observations, and $r_t = \widetilde{H}_t / \overline{H}_{\text{open}}$. Then

$$H(o_t) = -\sum_c p_t(c) \log_2 p_t(c),$$
$$s_{\text{OEC}} = \text{clamp}\Big(1 - \min_t r_t, 0, 1\Big).$$

Lower relative entropy indicates that observations have concentrated around a smaller body of evidence. Since an unproductive loop can produce the same pattern, OEC is used with TTE and DPI.

(A2) Progressive Scope Narrowing (PSN). We count the distinct files touched in a five-step window and compute Kendall's τ_K (Kendall, 1938) between this scope and the step index over the second half of the trajectory:

$$\text{scope}_t = \left| \bigcup_{k=\max(0,t-4)}^{t} F_k \right|,$$
$$s_{\text{PSN}} = \text{clamp}\left(\frac{1 - \tau_K}{2}, 0, 1\right).$$

A shrinking active file set gives negative τ_K and therefore a higher score, matching the localization pattern used by coding agents (Xia et al., 2024a; Yu et al., 2025b).

(B1) Intent–Action Consistency (IAC). A fixed lexicon-based parser maps the thought preceding each non-`think` action to a coarse intended action $\hat{a}_t$: reading, editing, execution/testing, or search. For the extractable steps $\mathcal{I} = \{t : \hat{a}_t \text{ is defined}\}$,

$$s_{\text{IAC}} = \frac{1}{|\mathcal{I}|} \sum_{t \in \mathcal{I}} \mathbf{1}[\hat{a}_t = a_t].$$

IAC is undefined when $\mathcal{I} = \varnothing$. It captures coarse thought–action consistency characteristic of ReAct-style interaction (Yao et al., 2023), without assessing semantic correctness.

(C1) Tool Transition Entropy (TTE). Let $p(i \to j)$ be the empirical frequency of adjacent transition (i, j) in the seven-action alphabet. We normalize its entropy by the maximum over 49 transition types:

$$s_{\text{TTE}} = \frac{-\sum_{i,j} p(i \to j) \log_2 p(i \to j)}{\log_2 49}.$$

TTE penalizes long single-tool loops. Focus and guardrail measures provide the context needed to distinguish useful variation from erratic switching.

(C2) Solution Commitment Point (SCP). A substantive edit is a `write_file` action whose observation contains no error pattern. Let t^* be the first such edit, $u = t^*/n$, $d(u) = \max(0, 0.2 - u, u - 0.5)$, and $\sigma = 0.2$. We set

$$s_{\text{SCP}} = \begin{cases} 0, & \text{no substantive edit;} \\ \exp\big(-d(u)^2/(2\sigma^2)\big), & \text{otherwise.} \end{cases}$$

The score is one for $u \in [0.2, 0.5]$ and decays for earlier or later commitment. Because u is relative time, SCP does not prefer a particular absolute trajectory length.

(C3) Post-Error Diversity (PED). Let $\mathcal{E}_\tau = \{t : \varepsilon_t = 1, t < n - 1\}$ be the error steps followed by an action. We combine a change in action type with the Jaccard distance between successive file sets:

$$\text{Jacc}(F_{t+1}, F_t) = 1 - \frac{|F_{t+1} \cap F_t|}{|F_{t+1} \cup F_t|},$$
$$d_t = \tfrac{1}{2}(\mathbf{1}[a_{t+1} \neq a_t] + \text{Jacc}(F_{t+1}, F_t)),$$
$$s_{\text{PED}} = \frac{1}{|\mathcal{E}_\tau|} \sum_{t \in \mathcal{E}_\tau} d_t,$$

PED is undefined when $\mathcal{E}_\tau$ is empty. Repeating the same action on the same files gives zero; changing the tool, file scope, or both raises the score (Shinn et al., 2023; Kamoi et al., 2024).

(D1) Dirty Pattern Indicator (DPI). DPI captures categorical failures that may survive the continuous measures: early stopping, no substantive edit, repeated shell-command loops, and repeated errors. With repeated-loop fraction *lp* and error-repetition rate *er*,

$$p = 0.4\,\mathbf{1}[\text{early}] + 0.4\,\mathbf{1}[\text{no edit}]$$
$$+ \min(0.3, 3\,\text{lp}) + \min(0.3, 0.6\,\text{er}),$$
$$s_{\text{DPI}} = 1 - \min(1, p).$$

A trajectory with no flagged pattern receives full guardrail credit; each detected pattern reduces the contribution of DPI.

Aggregation. We use fixed, hand-set weights throughout: $w_{\text{OEC}} = 0.30$, $w_{\text{IAC}} = 0.25$, $w_{\text{PSN}} = w_{\text{TTE}} = w_{\text{PED}} = w_{\text{DPI}} = 0.10$, and $w_{\text{SCP}} = 0.05$. The resulting totals are 0.40 for information focus, 0.25 for intent–action alignment, 0.25 for policy dynamics, and 0.10 for the guardrail. Perturbing w_{OEC}, w_{IAC}, or w_{DPI} one at a time retains 93.1–99.6% of the Top-1,024 selection (Appendix I).

The final TQS is a renormalized weighted mean over the components defined for a trajectory:

$$\text{TQS}(\tau) = \frac{\sum_{c \in \mathcal{C}^*(\tau)} w_c \cdot s_c(\tau)}{\sum_{c \in \mathcal{C}^*(\tau)} w_c}, \quad (1)$$

where $\mathcal{C}^*(\tau)$ is the set of components defined on τ. Trajectories with $n < 3$ or component coverage below $4/7$ are excluded. Outcome labels are used only for component admission during metric development (as described in §3.4); after the components and weights are fixed, TQS ranks successful trajectories using event-log structure alone.

3.4 Discriminant Validity and External Sanity Checks

We use task outcome as a weak admission signal for candidate components. Ineffective process patterns often make failure more likely, so a useful component should tend to rank a passing trajectory above a failing trajectory from the same task. This comparison filters signals with the wrong direction or strong confounds. Once the components are fixed, TQS ranks only task-successful trajectories for training, using event-log structure alone and no further patch-level supervision.

Concretely, we pair each passing trajectory with a failing trajectory from the same task. For each candidate measure, we compute two quantities. The first is concordance: the rate at which the measure assigns a higher score to the passing trajectory than to its paired failing trajectory, with 0.5 corresponding to chance. The second is the absolute Spearman correlation $|\rho|$ with trajectory length, which serves as a simple check against length-driven proxies. A measure is admitted only if its concordance exceeds 0.5 and $|\rho| < 0.25$.

Table 1 summarizes the results. The seven retained components pass both criteria, and the composite TQS reaches a concordance of 0.732, above every individual component. The same criteria exclude several candidates: BUS, TGCR, and DCS-file fail the concordance criterion, while DCS-Jaccard is dominated by length correlation ($|\rho| = 0.729$). HCR passes the numerical thresholds, but including it caused the Top-1K set to concentrate 99.9% of its mass on a single binary behavior pattern. This mode collapse is invisible to pairwise concordance but matters for SFT, where trajectories should preserve behavioral diversity. We therefore add behavioral-diversity preservation as a third admission criterion and exclude HCR under it. Additional details on the rejected measures are provided in Appendix E.

Measure	Conc.	$\lvert\rho\rvert$	Outcome
OEC	0.640	0.103	✓kept
IAC	0.612	0.087	✓kept
PSN	0.585	0.142	✓kept
SCP	0.598	0.058	✓kept
TTE	0.563	0.218	✓kept
PED	0.541	0.034	✓kept
DPI	0.591	0.071	✓guardrail
TQS Composite	**0.732**	0.092	—
HCR	0.527	0.014	× diversity
BUS/TGCR/DCS-f.	≤ 0.479	< 0.25	× conc.
DCS-Jacc.	—	0.729	× length

Table 1: Discriminant validity on paired success/failure trajectories from CoderForge-Preview. TQS's composite concordance exceeds every individual component. Dropped measures reflect the role of admission protocol: HCR is rejected for behavioral mode collapse, BUS/TGCR/DCS-file for low concordance, and DCS-Jaccard for length correlation.

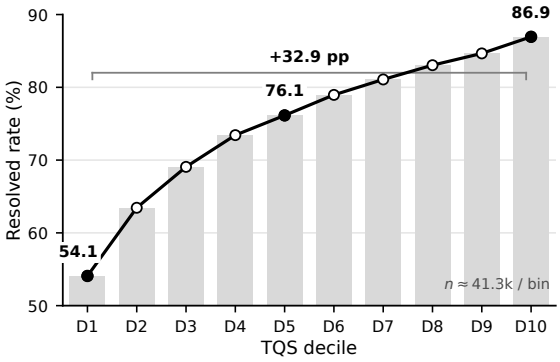


Figure 2: External validity of TQS on the released scoring records. We bin 413,278 records into ten equal-frequency deciles by TQS. These records include the source shards and the release's reward-1 convenience shards. The resolved rate increases monotonically from 54.1% in D1 to 86.9% in D10.

External outcome check. We also compare the finalized score with the test harness's `resolved` label, which is absent from the scoring function. Figure 2 shows a monotonic rise in resolved rate across the ten TQS deciles, from 54.1% in D1 to

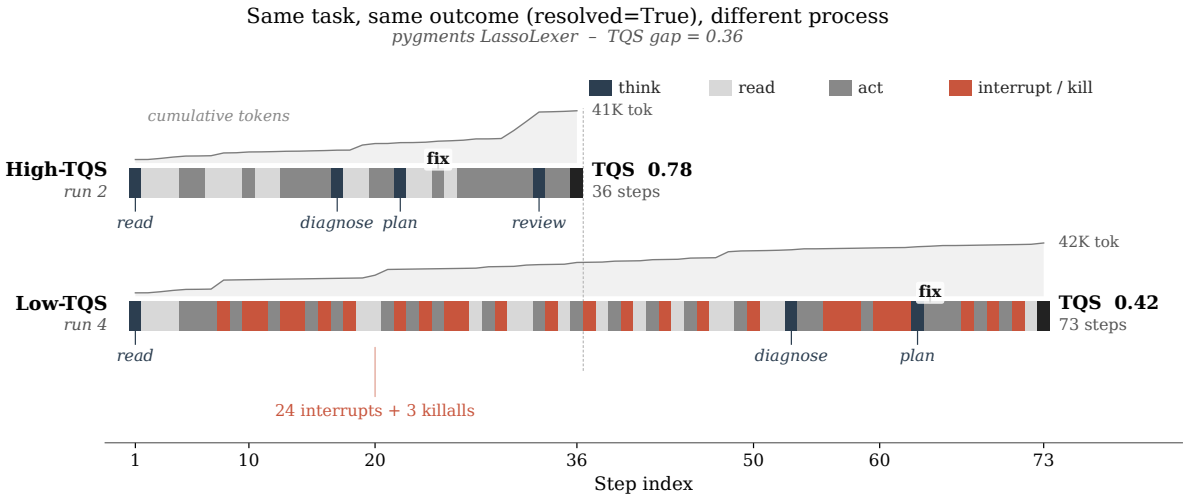


Figure 3: Two passing trajectories on the same `pygments` `LassoLexer` task. The high-TQS run reaches the fix in 36 steps. The low-TQS run takes 73 steps and includes many interrupts and forced process terminations. Their final outcomes are equal, and their TQS values differ by 0.36.

86.9% in D10. The 32.9-point spread has no inversion at a decile boundary. Within a fixed step-count band, pair-AUC is 1.000 for the composite and above 0.96 for OEC, IAC, and PED (Appendix G).

4 Experimental Setup

Qualitative motivation. Figure 3 shows two passing OpenHands runs on the same `pygments` `LassoLexer` task. Both resolve the task. The high-TQS trajectory reaches the fix in 36 steps through a compact read–diagnose–plan–review pattern; the low-TQS trajectory takes 73 steps and contains many interrupt and kill actions. The largest component gaps occur in observation focus (OEC: 0.738 vs. 0.312), commitment timing (SCP: 0.990 vs. 0.528), and post-error adaptation (PED: 1.000 vs. 0.000), revealing substantial behavioral differences that the final outcome alone does not capture. Appendix A reports all seven component scores.

Subset	Top	Bottom	Random
# trajectories	1,024	1,024	1,024
Mean TQS	0.714	0.454	0.589
Mean steps	34.2	53.0	41.0
Mean tokens (K)	28.1	26.3	26.9
Tasks / repos	847/209	847/204	825/217
Max per task	30	30	59
Source mix (%)	61/20/18	61/20/18	61/20/18
Dirty flag (%)	1.0	64.3	13.6

Table 2: Training subsets. Top and Bottom have zero TQS overlap (Top.min $= 0.706 >$ Bottom.max $= 0.470$). Source mix is Smith/Reb./Gym.

Data and subset construction. We score CoderForge-Preview (Ariyak et al., 2026), an open-license corpus of **258,134** unique OpenHands-format trajectories produced by a Qwen3-Coder-480B teacher on **51,201** tasks from R2E-Gym (Jain et al., 2025), SWE-Smith (Yang et al., 2025b), and SWE-Rebench (Badertdinov et al., 2025). The **155,144** passing trajectories form the candidate pool and were deduplicated by `trajectory_id` before subset construction. For fixed-budget SFT, we keep trajectories with at least 3 steps, TQS coverage $\geq 4/7$, and tokenized length at most 32,768, leaving **38,382** eligible trajectories. From this pool we construct matched Top-K, Bottom-K, and Random-K subsets of 1,024 trajectories, controlling task coverage, repository coverage, and source proportions. Table 2 summarizes the resulting subsets. Sampler details are in Appendix C.1.

Training and evaluation. All SFT runs use Qwen3-Coder-30B-A3B-Instruct (Yang et al., 2025a) with LoRA (Hu et al., 2022). The training recipe is fixed across conditions: adapters are placed on the attention projections, loss is applied to `assistant` and `tool_call` segments only, and every hyperparameter except the selected trajectories is held constant (Appendix C). We evaluate pass@1 on SWE-Bench Verified (OpenAI, 2024) using OpenHands v0.52.1 (Wang et al., 2025), matching the scaffold and version used to generate the training trajectories. We also recompute TQS components on evaluation rollouts to diagnose process-level behavioral changes.

ID	Method	$\lvert\mathcal{D}\rvert$	pass@1 by difficulty (%) Easy	Medium	Hard	Overall	ΔBase
No-SFT control							
B0	Qwen3-Coder-30B-A3B (base, no SFT)	—	36.6	19.2	2.2	24.4	—
Subset baselines (matched 1,024 trajectories, identical SFT recipe)							
B1	Random-K	1,024	40.7	16.9	2.2	24.8	+0.4
B2	Bottom-K (lowest TQS)	1,024	33.5	16.9	0.0	21.8	−2.6
B3	Shortest-K (length-only)	1,024	37.6	18.4	0.0	24.2	−0.2
B4	DPI-only-K (guardrail only)	1,024	37.6	19.9	2.2	25.2	+0.8
B5	LLM-Judge-K (GPT-4o rubric)	1,024	**43.3**	**21.1**	**6.7**	**28.4**	**+4.0**
B6	Top-LM-K (length-matched to Random)	1,024	40.7	20.7	4.4	27.0	+2.6
Ours	**Top-K (highest TQS)**	1,024	42.3	20.3	**6.7**	27.6	+3.2

Table 3: **Main results on SWE-Bench Verified.** All SFT runs use the same Qwen3-Coder-30B-A3B-Instruct base, LoRA hyperparameters, and OpenHands evaluation scaffold; only the selected training trajectories differ. Difficulty buckets follow OpenAI (2024).

Baselines. We compare against the untuned base model, Random-K, Bottom-K, Shortest-K by token count, DPI-only-K, GPT-4o LLM-Judge-K under an AgentDiagnose-style (Ou et al., 2025) rubric, and Top-LM-K, which length-matches high-TQS trajectories to the Random-K length distribution. All baselines are drawn from the same eligible pool and trained with the same SFT recipe.

5 Results

5.1 Main Results

Table 3 reports the main results. Under the same backbone, SFT recipe, and evaluation scaffold, the selected trajectories have a clear effect on downstream performance. Random-K improves only slightly over the base model, from 24.4 to 24.8 pass@1. In contrast, TQS-Top reaches 27.6, improving over Random-K by 2.8 points and over the base model by 3.2 points. Bottom-K moves in the opposite direction, dropping to 21.8. This ordering shows that trajectories with the same reward-1 outcome can differ substantially in their value as supervision.

The simpler baselines do not explain the TQS gain. Shortest-K reaches 24.2, below Random-K, so selecting short trajectories alone is not sufficient as a selection criterion. DPI-only-K gives a small improvement over Random-K but remains 2.4 points below TQS-Top. The length-matched control, Top-LM-K, reaches 27.0 after matching the length distribution of Random-K. It retains most of the Top–Random gain, which indicates that length is correlated with trajectory quality but does not fully account for the effect of TQS.

Among the non-TQS selectors, LLM-Judge-K is the strongest baseline at 28.4 pass@1. This is 0.8 points above TQS-Top, showing that rubric-based GPT-4o judging remains competitive. Still, TQS-Top recovers most of its improvement over Random-K and matches LLM-Judge-K on Hard tasks at 6.7. Figure 4 summarizes this trade-off between accuracy and selection cost.

Their selection costs differ more sharply. Under the prompt, token-count, and pricing assumptions used for the baseline, GPT-4o judging is estimated to cost $\sim\$2{,}750$ for the 38,382 eligible trajectories and $\sim\$18.6K$ for all 258,134 unique trajectories (Appendix F). In comparison, the current TQS implementation processes a 3,462-trajectory benchmark (median 53 steps) at 43 ms per trajectory with one worker on an AMD EPYC 7413. Extrapolating this measured rate gives about 3.1 CPU-hours for the unique corpus. With four workers, the measured throughput is 87.5 trajectories per second, corresponding to about 49 minutes for a full-corpus pass, or roughly \$0.1–\$0.2 in on-demand cloud CPU time at representative 4-vCPU rates (Appendix C). These measurements make TQS suitable for corpus-scale ranking, while semantic judging can be reserved for settings where its additional cost is acceptable.

The benefit of TQS selection is larger on harder tasks. The Top–Random gap increases from 1.6 points on Easy to 3.4 on Medium and 4.5 on Hard. This pattern suggests that generic successful supervision may suffice for easier problems, while more difficult tasks are more sensitive to the structure and quality of the selected trajectories.

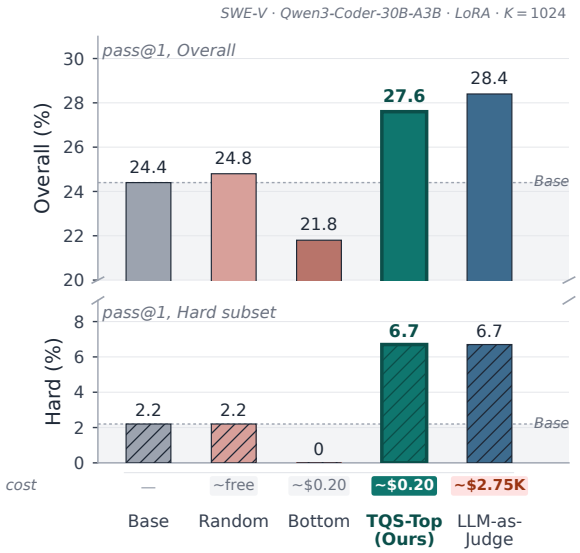


Figure 4: **Main results on SWE-Bench Verified.** Pass@1 on the full test set (top) and Hard subset (bottom). TQS-Top recovers most of the LLM-as-Judge gain at much lower selection cost.

Validation statistic	Human	Sonnet-5
High vs. Low agreement	85.2 (208/244)	81.1 (198/244)
Mid vs. Low agreement	74.2 (190/256)	77.0 (197/256)
High vs. Mid agreement	76.8 (172/224)	67.4 (151/224)
Overall agreement	**78.7** (570/724)	75.4 (546/724)
Pooled concordance	73.3 (631/861)	**76.2** (656/861)
Full-order match	**45.7** (137/300)	41.3 (124/300)
Spearman ρ	**0.417**	0.402

Table 4: Direct validation of the TQS ordering among successful runs. Agreement (%) uses order-consistent pairwise judgments; pooled concordance uses pointwise rubric scores with ties excluded.

5.2 Direct Validation among Successful Runs

We next test whether the TQS ordering agrees with content-based judgments while holding both task and outcome fixed. For each of 300 tasks with at least three resolved runs, we construct high–low, mid–low, and high–mid pairs, yielding 900 comparisons. Five software-engineering students split the human evaluation, with each pair assigned to one student, while Claude Sonnet-5 evaluates the same pairs separately. Both evaluations are blinded to TQS and its components. Each pair is presented in both orders; 176 pairs receive inconsistent verdicts and are treated as ties, leaving 724 order-consistent comparisons.

Table 4 shows consistent agreement across the three validation views. On the 724 order-consistent pairs, human and Sonnet-5 judgments agree with the TQS ordering on 78.7% and 75.4% of comparisons, respectively, with the strongest agreement on high–low pairs (85.2% and 81.1%). Pointwise rubric scores yield similar pooled concordance, at 73.3% for humans and 76.2% for Sonnet-5. More stringently, the complete high>mid>low ordering matches TQS on 45.7% and 41.3% of tasks, well above the 16.7% chance rate for three-item rankings. Together, these results show that TQS captures a substantial portion of the quality differences recognized by content-based evaluators, while remaining deliberately narrower than a semantic judge. Appendix B reports the full statistics and examples of semantic distinctions not captured by TQS.

Selection signal	pass@1
Full TQS-Top	**27.6**
w/o Information focus	24.6
w/o Intent–action alignment	23.0
w/o Policy dynamics	25.8
Random-K	24.8
Base (no SFT)	24.4

Table 5: Dimension ablation of TQS. Each variant removes one structural dimension from data selection and retrains with the same SFT recipe. Random-K and the untuned base model are shown for reference.

5.3 Ablation of TQS Dimensions

We next test whether TQS's downstream benefit comes from one structural dimension or their combination. From the full scorer, we remove information focus, intent–action alignment, or policy dynamics from data selection, reconstruct the Top-K subset, and retrain with the same SFT recipe.

Table 5 shows that all three dimensions contribute to the downstream gain. Removing intent–action alignment produces the largest drop, from 27.6 to 23.0 pass@1, falling below both Random-K and the untuned base model. Removing information focus reduces performance to 24.6, while removing policy dynamics gives 25.8. None of the ablated variants matches the full TQS result, indicating that the gain is not carried by a single dimension. The particularly large drop without intent–action alignment suggests that selecting trajectories whose stated reasoning is consistently reflected in subsequent actions is especially important for effective SFT supervision.

Process metric	Training Data	Fine-tuned models		
	Top-K	Top	Rand	Bottom
Mean Steps	34.2	**50.66**	57.64	63.46
Mean OEC	0.603	**0.571**	0.521	0.484
Mean PSN	0.753	**0.570**	0.548	0.568
Mean SCP	0.994	0.777	**0.798**	0.772
Mean PED	0.826	0.192	**0.267**	0.164
Mean DPI	0.996	**0.931**	0.923	0.906

Table 6: Process metrics of the Top-K training data and the fine-tuned models' own SWE-Bench Verified rollouts.

5.4 Behavioral Diagnosis on Rollouts

Finally, we ask whether data selection is reflected in the models' own test-time behavior. We recompute the TQS components on their SWE-Bench Verified rollouts and report the means in Table 6. The Top model uses 50.66 steps on average, compared with 57.64 for Random and 63.46 for Bottom. It also has higher OEC and DPI than both models; its PSN is higher than Random and close to Bottom. The pattern is not uniform across all components in these rollouts, since Random has slightly higher SCP and PED than Top. This suggests that selecting high-TQS supervision leaves a measurable behavioral imprint on the fine-tuned policy rather than affecting only final task accuracy. At the same time, the mismatch on SCP and PED indicates that the model does not simply reproduce the component profile of its training data. The clearest rollout differences are shorter interaction, more focused observations, and fewer dirty patterns.

6 Conclusion

"Passing" is too coarse a quality signal for coding-agent SFT data. We introduced TQS, a lightweight event-log scorer that evaluates successful trajectories through information focus, intent–action alignment, policy dynamics, and a guardrail for categorical failure patterns. Across our experiments, high-TQS trajectories provide more effective supervision than simpler outcome-, length-, or pattern-based selection, while low-TQS trajectories can even hurt downstream performance. At the same time, TQS approaches the effectiveness of rubric-based GPT-4o judging at substantially lower selection cost, making repeated rescoring and corpus-scale data selection practical. These results suggest that successful trajectories are not equally valuable as supervision, and that trajectory structure offers a useful, scalable signal beyond task outcome alone.

Limitations

Our experiments use CoderForge-Preview trajectories generated by a single teacher model under the OpenHands scaffold. Although the components rely on general event-log fields, their parsers and thresholds may require recalibration for different action spaces, multilingual traces, or other step granularities. Transfer to GUI or multimodal agents would also require new definitions of scope and intent. Outside the current setting, the hand-set component weights may likewise require recalibration. TQS also scores the structure of an observed process rather than verifying whether a passing patch addresses the underlying root cause.

Our SFT experiments focus on a fixed-budget setting with 1,024 selected trajectories per condition. We also restrict training trajectories to at most 32K tokens so that they fit within the 40,960-token context after template overhead. This excludes a long-trajectory tail that may contain useful recovery behavior. We train each condition once because of compute cost, so the reported pass@1 differences do not include multi-seed variance estimates.

Our downstream evaluation is limited in model scale. Preliminary experiments with a 7B backbone did not show stable selector differences, so we focus the reported SFT comparison on the 30B model. Larger backbones remain untested.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. This work was supported by the Fundamental Research Funds for the Central Universities, Peking University. Xu Sun is the corresponding author of this paper.

References

Alpay Ariyak, Junda Zhang, Junxiong Wang, Shang Zhu, Federico Bianchi, Sanjana Srivastava, Ashwinee Panda, Siddhant Bharti, Chenfeng Xu, John Heo, Xiaoxia Shirley Wu, James Zou, Percy Liang, Leon Song, Ce Zhang, Ben Athiwaratkun, Zhongzhu Zhou, and Qingyang Wu. 2026. CoderForge-Preview: SOTA open dataset for training efficient coding agents. Dataset card and technical release.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. 2025. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *Preprint*, arXiv:2505.20411.

Oliver Bentham, Nathan Stringham, and Ana Marasović. 2024. Chain-of-thought unfaithfulness as disguised accuracy. *Preprint*, arXiv:2402.14897.

Guhong Chen, Chenghao Sun, Cheng Fu, Qiyao Wang, Zhihong Huang, Chaopeng Wei, Guangxu Chen, Feiteng Fang, Ahmadreza Argha, Bing Zhao, Xander Xu, Qi Han, Hamid Alinejad-Rokny, Qiang Qu, Binhua Li, Shiwen Ni, Min Yang, Hu Wei, and Yongbin Li. 2026. Beyond quantity: Trajectory diversity scaling for code agents. *Preprint*, arXiv:2602.03219.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2024. AlpaGasus: Training a better alpaca with fewer data. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. 2025. Agentic reinforced policy optimization (ARPO). *Preprint*, arXiv:2507.19849.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. 2025. R2E-Gym: Procedural environments and hybrid verifiers for scaling open-weights SWE agents. *Preprint*, arXiv:2504.07164.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world GitHub issues? In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can LLMs actually correct their own mistakes? A critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Maurice G. Kendall. 1938. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamilė Lukošiūtė, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Larson, Sam McCandlish, Sandipan Kundu, and 11 others. 2023. Measuring faithfulness in chain-of-thought reasoning. *Preprint*, arXiv:2307.13702.

Ming Li, Yong Zhang, Shwai He, Zhitao Li, Hongyu Zhao, Jianzong Wang, Ning Cheng, and Tianyi Zhou. 2024a. Superfiltering: Weak-to-strong data filtering for fast instruction-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2024b. From quantity to quality: Boosting LLM performance with self-guided data selection for instruction tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Zeping Li, Hongru Wang, Yiwen Zhao, Guanhua Chen, Yixia Li, Keyang Chen, Yixin Cao, Guangnan Ye, Hongfeng Chai, Mengdi Wang, and Zhenfei Yin. 2026. Rethinking the role of entropy in optimizing tool-use behaviors for large language model agents. *Preprint*, arXiv:2602.02050.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. 2024. What makes good data for alignment? A comprehensive study of automatic data selection in instruction tuning. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

Murong Ma, Tianyu Chen, Yun Lin, Shuai Lu, Qinglin Zhu, Yeyun Gong, Zhiyong Huang, Peng Cheng, Yan Lu, and Jin Song Dong. 2026. From patches to trajectories: Privileged process supervision for software-engineering agents. *Preprint*, arXiv:2605.21996.

Yingwei Ma, Rongyu Cao, Yongchang Cao, Yue Zhang, Jue Chen, Yibo Liu, Yuchen Liu, Binhua Li, Fei Huang, and Yongbin Li. 2024. Lingma SWE-GPT: An open development-process-centric language model for automated software improvement. *Preprint*, arXiv:2411.00622.

OpenAI. 2024. Introducing SWE-bench Verified. OpenAI Blog.

Tianyue Ou, Wanyao Guo, Apurva Gandhi, Graham Neubig, and Xiang Yue. 2025. AgentDiagnose: An open toolkit for diagnosing LLM agent trajectories. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP Demos)*, pages 207–215.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2025. Training software engineering agents and verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*.

Claude E. Shannon. 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. 2025. OS-Genesis: Automating GUI agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5555–5579.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2025. OpenHands: An open platform for AI software developers as generalist agents. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*.

Yejie Wang, Keqing He, Dayuan Fu, Zhuoma GongQue, Heyang Xu, Yanxu Chen, Zhexu Wang, Yujia Fu, Guanting Dong, Muxi Diao, Jingang Wang, Mengdi Zhang, Xunliang Cai, and Weiran Xu. 2024. How do your code LLMs perform? empowering code instruction tuning with really good data. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14027–14043, Miami, Florida, USA. Association for Computational Linguistics.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. 2025. SWE-RL: Advancing LLM reasoning via reinforcement learning on open software evolution. *Preprint*, arXiv:2502.18449.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024a. Agentless: Demystifying LLM-based software engineering agents. *Preprint*, arXiv:2407.01489.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. 2024b. LESS: Selecting influential data for targeted instruction tuning. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 2 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.

Jiahao Yu, Zelei Cheng, Xian Wu, and Xinyu Xing. 2025a. Building coding agents via entropy-enhanced multi-turn preference optimization. *Preprint*, arXiv:2509.12434.

Zhongming Yu, Hejia Zhang, Yujie Zhao, Hanxian Huang, Matrix Yao, Ke Ding, and Jishen Zhao. 2025b. OrcaLoca: An LLM agent framework for software issue localization. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pages 73416–73436.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. LIMA: Less is more for alignment. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*.

A Worked-Example Component Scores

Table 7 gives the component breakdown for the two `pygments` trajectories in Figure 3. The table uses the fixed weights defined in §3.3.

B Direct Validation and Semantic Gaps

Within-task protocol. We sample 300 tasks with at least three resolved runs: 181 from SWE-Smith, 85 from SWE-Rebench, and 34 from R2E-Gym. High, middle, and low TQS tiers yield 900 pairs. Each pair is presented in both orders, giving 1,800 judgments in the human study and another 1,800 in the Claude Sonnet-5 study. Five software-engineering students split the human labeling, with one student assigned to each pair. Neither study sees TQS or its components. The rubric covers problem comprehension, solution quality and minimality, exploration efficiency, error recovery, and verification rigor. Of the 900 pairs, 176 reverse under presentation order and are treated as ties. The main table reports the remaining 724 pairs, or 80.4% coverage.

The rubric scores also give a pointwise comparison that does not require the two presentations to produce the same discrete verdict. Table 9 reports pairwise concordance after excluding equal-score ties. Across the three pair types, the

Component	Weight	Run 2	Run 4
OEC	0.30	0.7384	0.3120
IAC	0.25	0.6667	0.5610
PSN	0.10	0.8491	0.5708
TTE	0.10	0.5718	0.4046
SCP	0.05	0.9904	0.5280
PED	0.10	1.0000	0.0000
DPI	0.10	1.0000	0.6000
TQS		**0.7798**	**0.4178**

Table 7: Component scores for the two resolved runs in Figure 3. Run 2 is the high-TQS trajectory and Run 4 is the low-TQS trajectory.

human rubric agrees with TQS on 631/861 pairs and Claude Sonnet-5 agrees on 656/861. The full high>mid>low ordering matches TQS for 137/300 tasks under the human rubric and 124/300 tasks under Claude Sonnet-5, compared with $1/6$ of tasks under a random ordering.

What semantic judging adds. We inspect disagreements in which Claude Sonnet-5 prefers the lower-TQS trajectory. On `python__mypy-10430`, the preferred run traces a timing-dependent crash to a flag missing from `VAR_FLAGS` across serialization and fixes it in one line. The higher-TQS run duplicates lookup logic without resolving the timing failure. On the `sqlfluff func_pm_op_change` task, the preferred run corrects an indexing error, while the higher-TQS run handles a secondary tuple-subtraction failure and adds bounds checks around the original bug.

Verification produces another type of disagreement. On `wfondrie__mokapot-65`, both runs apply the same one-line fix. The preferred run builds a reproduction, executes the existing test, and adds edge-case tests. The higher-TQS run stops after static inspection without confirming a passing test. These examples show information available to semantic judging that TQS does not encode.

Score geometry and selection stability. TQS is a scalar functional, $\text{TQS} : \mathcal{T} \to [0, 1]$, rather than a structural distance between trajectories. Its role is to induce the total preorder used for subset selection: $\tau_i \succeq \tau_j$ whenever $\text{TQS}(\tau_i) \geq \text{TQS}(\tau_j)$. Distinct trajectories may therefore be equivalent for selection if they receive the same score. Accordingly,

$$d(\tau_i, \tau_j) = |\text{TQS}(\tau_i) - \text{TQS}(\tau_j)|$$

is a pseudometric on $\mathcal{T}$, and becomes a metric after identifying trajectories with equal scores. This only describes separation in score space; nearby TQS values do not imply structural similarity between trajectories.

For subset selection, the relevant quantity is the score margin. Let $s(\tau) = \text{TQS}(\tau)$ and let $\tilde{s}$ be a perturbed scoring function such that

$$\sup_{\tau \in \mathcal{T}} |\tilde{s}(\tau) - s(\tau)| \leq \epsilon.$$

Then, for any pair τ_i, τ_j,

$$\tilde{s}(\tau_i) - \tilde{s}(\tau_j) \geq s(\tau_i) - s(\tau_j) - 2\epsilon.$$

Hence, any pair whose original score gap exceeds 2ϵ preserves its ordering. For scores sorted as $s_{(1)} \geq \cdots \geq s_{(N)}$, define the Top-$K$ boundary margin as $\Delta_K = s_{(K)} - s_{(K+1)}$. If $\Delta_K > 2\epsilon$, the Top-K subset is unchanged. More generally, only trajectories close to the selection boundary can enter or leave the selected set under a bounded perturbation.

This also gives a simple robustness condition for the TQS weights. Writing

$$\text{TQS}_{\mathbf{w}}(\tau) = \sum_{m=1}^{M} w_m \phi_m(\tau), \qquad \phi_m(\tau) \in [0, 1],$$

a perturbation $\delta\mathbf{w}$ satisfies

$$|\text{TQS}_{\mathbf{w}+\delta\mathbf{w}}(\tau) - \text{TQS}_{\mathbf{w}}(\tau)| \leq \|\delta\mathbf{w}\|_1.$$

Therefore, $\Delta_K > 2\|\delta\mathbf{w}\|_1$ is a sufficient worst-case condition for preserving the Top-K subset. This bound is conservative, while the empirical Top-K overlap under weight perturbations directly measures the local stability of the selected subset around the chosen TQS configuration.

C Implementation Details

Parser. Trajectories are stored as OpenHands-format parquet files with the columns `trajectory_id`, `instance_id`, `repo`, `messages`, `tools`, `reward`, and `finish_reason`. Our parser converts each row into the step tuple in §3. Tool calls are recovered from the OpenAI-style `messages/tool_calls` schema, and observations are attached to the corresponding tool step whenever the event log contains a returned tool message. File paths are extracted from tool arguments and normalized before scope-based measures are computed.

	Human			Claude Sonnet-5		
Contrast	**Agreement**	**95% CI**	p	**Agreement**	**95% CI**	p
High vs. Low	85.2% (208/244)	80.3–89.1	1.34×10^{-30}	81.1% (198/244)	75.8–85.6	1.18×10^{-23}
Mid vs. Low	74.2% (190/256)	68.5–79.2	4.27×10^{-15}	77.0% (197/256)	71.4–81.7	1.51×10^{-18}
High vs. Mid	76.8% (172/224)	70.8–81.8	3.43×10^{-16}	67.4% (151/224)	61.0–73.2	2.04×10^{-7}
Overall	**78.7% (570/724)**	75.6–81.6	5.94×10^{-57}	**75.4% (546/724)**	72.1–78.4	2.70×10^{-44}

Table 8: Agreement with the TQS ordering among order-consistent pairs. Confidence intervals are 95% Wilson intervals. The p-values use two-sided exact binomial tests against 0.5.

	Human rubric			Claude Sonnet-5 rubric			
Contrast	**Conc.**	**95% CI**	p	**Conc.**	**95% CI**	p	**Ties**
High vs. Low	78.3% (227/290)	73.2–82.6	6.52×10^{-23}	82.1% (238/290)	77.2–86.1	1.35×10^{-29}	10
Mid vs. Low	69.5% (203/292)	64.0–74.5	2.11×10^{-11}	75.7% (221/292)	70.5–80.3	4.39×10^{-19}	8
High vs. Mid	72.0% (201/279)	66.5–77.0	1.11×10^{-13}	70.6% (197/279)	65.0–75.6	4.39×10^{-12}	21
Pooled	**73.3% (631/861)**	70.2–76.1	6.59×10^{-44}	**76.2% (656/861)**	73.2–78.9	1.05×10^{-55}	39

Table 9: Concordance between pointwise rubric scores and the TQS tier ordering after excluding equal-score ties. Confidence intervals are 95% Wilson intervals; p-values use two-sided exact binomial tests against 0.5.

Scoring cost. The scorer is a deterministic CPU pipeline, and each component is linear in trajectory length. We benchmark the current implementation on an AMD EPYC 7413 using 3,462 mixed-source trajectories with a median length of 53 steps. A single worker processes this batch in 149.2 seconds, or 23.2 trajectories per second (43 ms per trajectory). At this rate, scoring all 258,134 unique trajectories requires about 3.1 CPU-hours. Four workers reach 87.5 trajectories per second, which corresponds to about 49 minutes for a full-corpus pass. Profiling attributes 58% of the single-worker runtime to regular-expression error detection and 13% to the character-level entropy calculation used by OEC.

Loss masking. For SFT, ms-swift computes loss on assistant responses, including tool calls, while masking user messages and tool responses. Under the Qwen3-Coder agent template, tool calls are rendered in XML. For example:

```
<tool_call>
<function=execute_bash>
<parameter=command>
pytest tests/
</parameter>
</function>
</tool_call>
```

This setup trains the model to imitate both natural-language reasoning and tool invocation, while excluding user messages and tool observations from the supervised loss. As a result, the loss focuses on the agent's reasoning and actions, while tool outputs serve only as conditioning context.

Error flags. The binary error flag ε_t is set from tool observations using a small set of string and regular-expression patterns, including common execution failures, missing-file messages, traceback markers, failed test invocations, and tool-level error statuses. The flag is intentionally conservative: it is used to detect repeated failure patterns and post-error behavior, not to infer semantic correctness of the final patch.

C.1 Subset Sampler

We construct fixed-budget training subsets from the eligible pool described in §3.3. Each subset contains 1,024 trajectories. Top-K and Bottom-K are selected by descending and ascending TQS, respectively, while Random-K is sampled with a fixed seed. To keep the comparison focused on trajectory quality, the sampler controls three sources of variation: task identity, repository coverage, and source provenance.

For Top-K and Bottom-K, we first sort eligible trajectories by TQS in the corresponding direction. The sampler then fills the subset greedily. In the first pass, it selects at most one trajectory per task. In the second pass, it prioritizes trajectories from repositories not yet represented in the subset. In the final pass, it fills any remaining slots in score order, subject to a hard per-task cap of 30 trajectories. This preserves TQS ranking as the primary selection criterion while avoiding subsets dominated by a small number of tasks or repositories.

Because CoderForge-Preview combines trajectories from multiple data sources, we also stratify

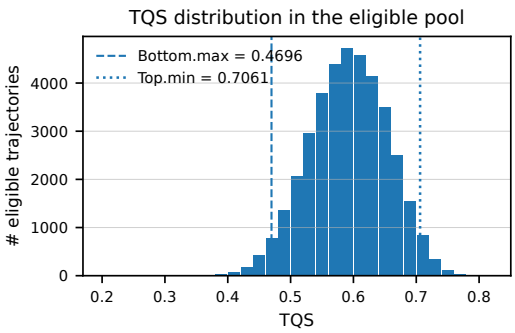


Figure 5: TQS distribution on the 38,382-trajectory eligible pool. The distribution peaks around 0.58–0.60. The Top-1024 threshold is 0.7061, and the Bottom-1024 threshold is 0.4696, leaving a gap of 0.2365 between the two selected subsets.

Component	Mean	Std.	Undefined
OEC	0.429	0.118	0.0%
PSN	0.646	0.153	0.0%
IAC	0.570	0.098	0.0%
TTE	0.550	0.038	0.0%
SCP	0.978	0.060	0.0%
PED	0.580	0.232	0.7%
DPI	0.941	0.127	0.0%

Table 10: Marginal statistics of the seven retained TQS components on the eligible pool. SCP and DPI are concentrated near 1, while OEC, PSN, IAC, and PED provide broader variation.

each subset by source. Target source proportions are computed from the eligible pool and applied to Top-K, Bottom-K, and Random-K. Random-K uses the same eligibility filters and source proportions, but samples uniformly within each source stratum rather than sorting by TQS. All subset construction is performed after deduplication by `trajectory_id`. Ties at selection boundaries are broken deterministically by `trajectory_id`.

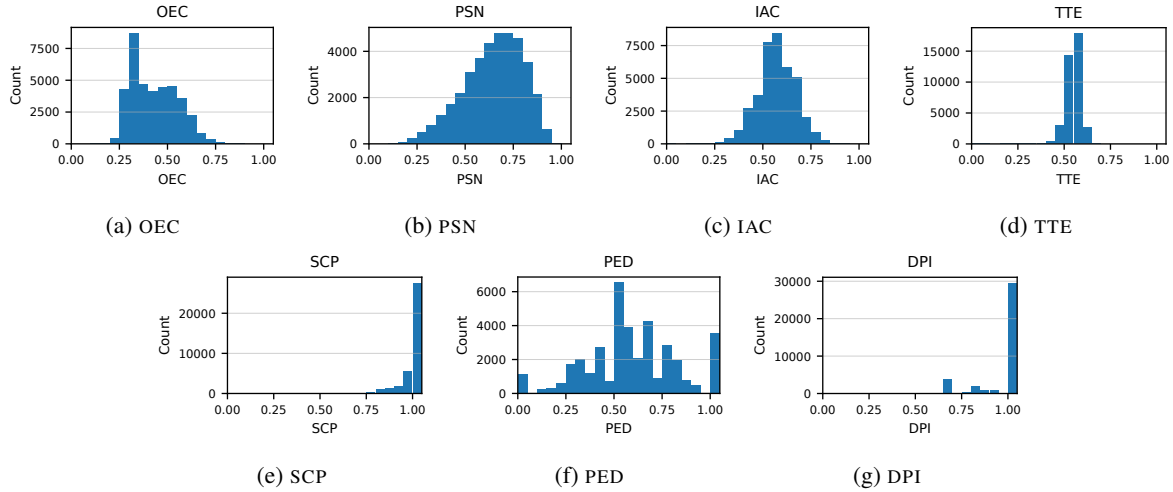
D Score and Component Distributions

Figure 5 shows the distribution of the final TQS over the 38,382 eligible trajectories. The scores span a broad range rather than collapsing into a narrow band, providing sufficient resolution for fixed-budget ranking. The Top-1,024 cutoff is 0.7061, whereas the Bottom-1,024 cutoff is 0.4696, leaving a score gap of 0.2365 between the two selected regions. Random-K, by construction, remains centered around the eligible-pool distribution rather than either tail. This separation is important operationally: TQS does not merely reorder trajectories with nearly identical scores, but identifies distinct high- and low-scoring regions from which fixed-size subsets can be constructed. The separation alone does not establish downstream quality, but provides the score resolution required for the selection experiments in §5.

Table 10 and Figure 6 characterize the seven retained components individually. Their marginal distributions differ substantially, consistent with their different roles in structured convergence. PED has the largest dispersion ($\sigma = 0.232$), followed by PSN ($\sigma = 0.153$) and OEC ($\sigma = 0.118$), providing substantial variation across the eligible pool. IAC also varies meaningfully around its mean of 0.570, whereas TTE is more tightly concentrated ($\sigma = 0.038$). In contrast, SCP and DPI have high means of 0.978 and 0.941, respectively, because most eligible trajectories already avoid their most severe failure patterns. Their value therefore lies primarily in identifying the smaller tail of runs with weak commitment or dirty interaction patterns rather than providing continuous discrimination across the entire pool.

The components also differ in coverage. Six are defined for every eligible trajectory, while PED is undefined for only 0.7% of runs because its error-recovery signal requires an applicable error transition. This small amount of missingness is handled by the renormalized aggregation in Eq. 1, rather than assigning an artificial value to trajectories for which the component cannot be evaluated. Overall, the heterogeneous marginals show that the final score combines both broadly varying signals such as OEC, PSN, and PED and more selective guardrail-like signals such as SCP and DPI.

Figure 7 further examines whether these signals are redundant. Pairwise Spearman correlations remain modest across the retained components; the largest off-diagonal correlation is only 0.320, between TTE and PED. Thus, no component is well approximated by another through a simple monotonic relationship. This is consistent with the intended design of TQS: information focus, intent–action alignment, policy dynamics, and categorical failure patterns contribute overlapping but distinct structural evidence. The correlation analysis by itself does not show that every component is necessary, but it rules out severe marginal redundancy; the downstream dimension ablations in §5.3 provide the complementary test of whether these signals matter for data selection.



(a) OEC (b) PSN (c) IAC (d) TTE

(e) SCP (f) PED (g) DPI

Figure 6: Marginal distributions of the seven retained TQS components on the eligible pool. All components are normalized to $[0, 1]$ before aggregation.

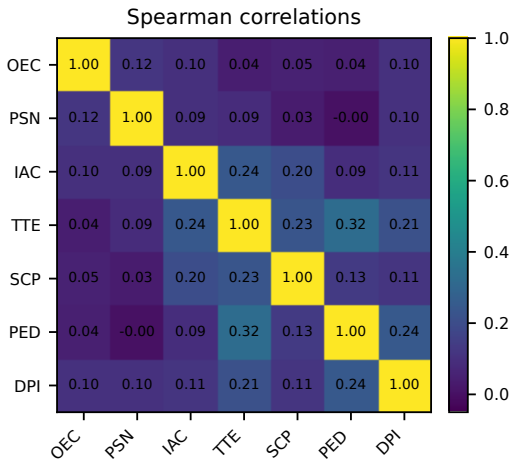


Figure 7: Spearman correlation matrix among retained TQS components on the eligible pool. The largest off-diagonal correlation is 0.320 between TTE and PED, indicating no severe redundancy among the retained components.

E Failed Candidate Measures

We also considered several candidate measures that were motivated by the same structured-convergence view but failed the admission protocol in §3.4. Their definitions are included here to clarify the boundary of the final score.

HCR (Hypothesis–Commit Ratio). HCR characterizes how an agent allocates steps between exploration and execution. We define exploration as $\mathcal{R} = \{\texttt{think}, \texttt{read_file}, \texttt{search}\}$ and execution as $\mathcal{E} = \{\texttt{write_file}, \texttt{execute_bash}\}$, and compute their ratio within the early, middle, and late thirds of the trajectory:

$$R_g = \frac{\#\{t \in G_g : a_t \in \mathcal{R}\}}{\#\{t \in G_g : a_t \in \mathcal{E}\}}.$$

HCR assigns score 1 when $R_{\text{early}} \geq R_{\text{mid}} \geq R_{\text{late}}$, favoring an explore-then-commit pattern, and 0 otherwise. Although it passes the numerical thresholds (concordance 0.527, $|\rho| = 0.014$), including HCR caused 99.9% of the Top-1K subset to share the same binary pattern, so we exclude it under the behavioral-diversity criterion.

BUS (Backtracking Utility Score). BUS distinguishes productive redirection from unproductive wandering. A backtracking event is detected when the agent's current target-file set has no overlap with that of the preceding k steps. The event is considered useful if, within the following k steps, the search scope narrows or the agent makes a substantive edit:

$$s_{\text{BUS}} = \frac{\#\text{ useful backtracking events}}{\#\text{ backtracking events}}.$$

Higher BUS therefore indicates that changes in direction more often lead to focused progress. Its concordance is only 0.202, so we exclude it.

TGCR (Tool-Graph Cycle Ratio). TGCR captures structural redundancy in file navigation. We construct a directed graph whose nodes are files and whose edges represent successive file-access transitions, and use DFS coloring to identify back edges:

$$s_{\text{TGCR}} = \frac{\#\text{ back edges}}{\#\text{ total edges}}.$$

A larger value indicates more cyclic revisitation, whereas a trajectory closer to a DAG reflects a cleaner access path. TGCR obtains concordance 0.479, close to chance, and is therefore excluded.

DCS-Jaccard (Decision Consistency Score). DCS measures local consistency between reasoning, action, and subsequent feedback. For each step, it combines intent–action alignment with the extent to which the resulting observation informs the next intent:

$$s_{\text{DCS}} = \frac{1}{2}\Big[\text{sim}(intent_t, action_t) + \text{sim}(obs_t, intent_{t+1})\Big].$$

DCS-Jaccard instantiates $\text{sim}(\cdot,\cdot)$ with Jaccard similarity and averages the resulting scores over valid steps. Although intended to capture coherent local transitions, it has $|\rho| = 0.729$ with step count and is therefore rejected as strongly length-confounded.

DCS-file (file-alignment variant). DCS-file replaces lexical Jaccard overlap with file-level alignment when evaluating the same intent–action and observation–intent transitions. It therefore asks whether the files implied by the agent's reasoning agree with those actually acted on and whether observed files are reflected in the next plan. The resulting score obtains concordance 0.238, substantially below chance in the intended direction, and is excluded.

F LLM-Judge Baseline Rubric

The B5 baseline uses GPT-4o as a trajectory quality judge under an AgentDiagnose-style rubric. The judge sees the issue description, the trajectory's reasoning and tool-call sequence, and the four rubric dimensions below. It does not see the reference patch or held-out test outcome. Each dimension is scored on a 1–5 Likert scale, and the trajectory score is the mean of the four dimensions. The top 1,024 trajectories by mean score, after the same eligibility filters and subset constraints as Top-K, form the B5 training subset.

Planning. Does the agent decompose the issue into a coherent plan before acting, and does the plan track the issue's structure rather than the first failure mode that comes to mind?

Score / Component	Pair-AUC
TQS composite	1.000
OEC	0.996
IAC	0.963
PED	0.961
PSN	0.839

Table 11: Length-controlled pair-AUC in the modal step-count band total_steps $\in [25, 40]$.

Grounding. Are claims and edits grounded in observations that the trajectory has actually surfaced, or does the agent assert facts about the codebase without prior evidence?

Reflection. When an action fails, does the agent diagnose the failure before retrying, or does it loop on minor variations of the same action?

Efficiency. Does each step advance the trajectory toward a fix, or does the trajectory contain redundant exploration, repeated reads of already-viewed files, or stuck-process interrupts?

Cost estimate. On the **38,382**-trajectory eligible pool, with approximately **28K** input tokens and **200** output tokens per call, GPT-4o pricing of \$2.50/M input tokens and \$10/M output tokens gives a per-pool cost of roughly $\sim\$2{,}750$. Applying the same judge to all **258,134** unique trajectories would cost roughly $\sim\$18.6K$.

G Length-Controlled Pair-AUC

Table 11 gives a stricter length-confound check than the $|\rho| < 0.25$ criterion in Table 1. We restrict trajectories to the modal step-count band, total_steps $\in [25, 40]$, and ask whether retained components still separate Top from Bottom. The composite TQS remains perfectly separating in this band. Among the largest components, OEC, IAC, and PED remain above 0.96 pair-AUC, while PSN drops to 0.839. This drop is diagnostic: scope narrowing is partly entangled with trajectory length, but the composite score is not driven by this single component.

H Tokenized-Length Cutoff

The 32,768-token cap is a packing-budget choice for SFT. It fits inside the 40,960-token context window while leaving room for chat-template and assistant-tag tokens. Of the **155,144** reward$= 1$ trajectories, **38,382** survive the eligibility filters and deduplication, giving a retention rate of **24.7%**.

Source	Passing	Eligible	Retention
SWE-Smith	89,501	23,569	26.3%
SWE-Rebench	44,739	7,821	17.5%
R2E-Gym	20,904	6,992	33.4%
All	155,144	38,382	24.7%

Table 12: Retention after applying the fixed-budget SFT eligibility filters to passing trajectories. All passing trajectories satisfy the step-count and coverage filters; the main reduction comes from the tokenized-length cap and subsequent deduplication by `trajectory_id`.

Weight setting	Top-1024 overlap
Main weights	100.0%
$w_{\text{OEC}} = 0.25$	93.1%
$w_{\text{OEC}} = 0.35$	93.6%
$w_{\text{IAC}} = 0.20$	93.1%
$w_{\text{IAC}} = 0.30$	94.1%
$w_{\text{DPI}} = 0.05$	98.3%
$w_{\text{DPI}} = 0.15$	99.6%

Table 13: Weight sensitivity of TQS selection on the eligible pool. We perturb one weight at a time while keeping the other weights fixed, then measure overlap between the resulting Top-1024 set and the main Top-1024 set.

The discarded tail is dominated by long trajectories, especially from SWE-Smith.

The cap should be interpreted as part of the fixed-budget SFT setting. Longer trajectories may contain useful recovery behavior, but they require longer-context or sequence-parallel training to imitate faithfully. We leave this setting to future work.

I Sensitivity to Component Weights

The main experiments use the fixed, hand-set weights defined in §3.3. They allocate 0.40 to information focus, 0.25 to intent–action alignment, 0.25 to policy dynamics, and 0.10 to the DPI guardrail. No downstream pass@1 result was used to tune them. We test local selection stability by perturbing the two largest individual weights, w_{OEC} and w_{IAC}, and the guardrail weight w_{DPI}. For each setting, we rebuild the Top-K subset and report its overlap with the main selection. This experiment measures how much the selected set changes near the chosen weights.

Figure 8 visualizes the same result. OEC and IAC perturbations retain 93–94% overlap with the main Top-1024 set, while DPI perturbations retain over 98% overlap. These results indicate that the selected subset is locally stable to moderate changes

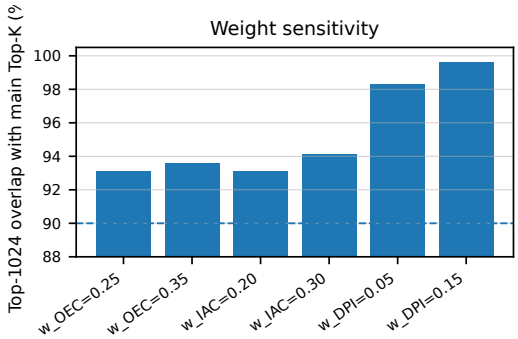


Figure 8: Weight sensitivity of Top-1024 selection under single-weight perturbations.

in the hand-set weights. This analysis does not establish that the chosen weights are globally optimal, but it shows that small perturbations around them leave most of the selected training set unchanged.