LatentGate: Policy-State Filtering for Token-Efficient Verification of Software Engineering Agents

Yuning Han
CISE Department
University of Florida
yuninghan@ufl.edu

Yangchenchen Jin
CISE Department
University of Florida
yangchenchen.jin@ufl.edu

Tyler Jandreau
Aeronix
tjandreau@aeronix.com

Jingwei Sun
CISE Department
University of Florida
sun.jingwei@ufl.edu

Abstract

Test-time scaling improves software engineering agents by generating multiple candidate trajectories per task, but selecting among them can incur a token cost comparable to that of candidate generation. In hybrid verification workflows, this execution-free (EF) verifier forms the first stage and is itself a major source of inference token cost. We introduce LatentGate, a token-free, execution-free filter that replaces this stage by reusing hidden states the policy already produces during generation. LatentGate represents each trajectory through three channels: chain-of-thought reasoning, environment observations, and function calls carrying concrete tool use and code edits. A contrastive branch compares each channel with banks of successful and unsuccessful experience and fuses the channels by within-candidate minimum rank, so a candidate scores highly only when every channel supports it. A lightweight linear scorer trained on the same policy states supplies a complementary signal, and the two are combined as equal-weight ranks. LatentGate involves no token cost or test execution to select the top candidates, and the following execution-based stages and the final language-model verifier only operate on half the candidate pool, which reduces the token cost of the entire verification workflow significantly. On SWE-bench Verified at $K = 16$, LatentGate reduces EF-verifier token cost by 78–79% across DeepSWE-Preview and R2EGym-32B and total verification tokens, including test generation, by 49–62%, while matching or slightly exceeding the original workflow.

1 Introduction

Efforts to improve LLM reasoning in academia and industry follow two broad directions: training more capable models to solve harder problems, and allocating more computation to each problem at inference time. Training can strengthen the model's reasoning ability before deployment [12]. The latter approach, known as test-time scaling, seeks better answers by allowing longer reasoning, generating multiple candidates, or searching with verifier feedback [2, 30, 39, 44, 45]. Rather than relying on a single attempt, these methods spend additional inference compute exploring and selecting solutions of the problem.

For software engineering (SWE) tasks, recent open agents use a hybrid verification workflow to select among generated trajectories [15, 26]. Figure 1 shows its four stages: an execution-free (EF) verifier scores all candidates and retains half; instance-supplied tests provide regression signals; LLM-generated tests provide additional execution signals; and the EF scores derived from Stage 1

Preprint. Under review.

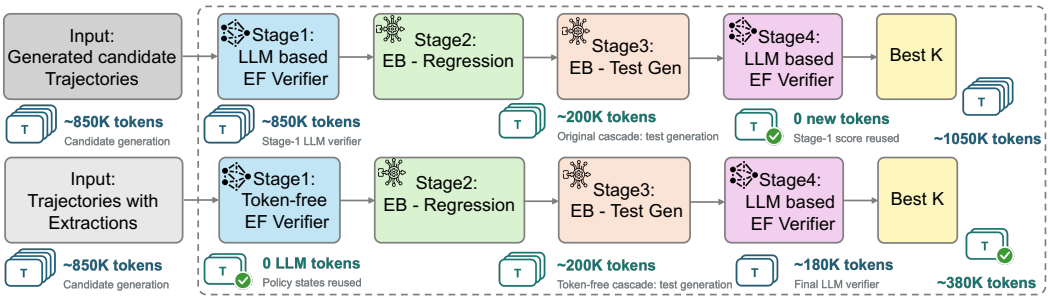

Figure 1: **Original and token-free hybrid verification pipelines.** Both systems select from the same pool of K generated candidate trajectories. The gray dashed boxes delimit inference-time verification, where our method reduces token consumption. In Stage 1, the original pipeline runs an LLM-based execution-free (EF) verifier, whereas ours uses the token-free EF verifier; each retains the top half of the candidates. Stage 2 executes the test cases supplied with each instance to obtain regression signals. Stage 3 uses an LLM-based execution-based (EB) verifier to generate and run additional test cases, yielding test-generation signals. In Stage 4, the original pipeline reuses its Stage 1 EF scores to select the final candidate, while ours applies the LLM-based EF verifier only to the remaining candidates. At $K = 16$ on DeepSWE-Preview, candidate generation costs about 850K tokens per issue. The original cascade then spends about 850K tokens in the Stage-1 LLM verifier and 200K in test generation, about 1050K in total, whereas ours spends no LLM tokens in Stage 1, the same 200K in test generation, and about 180K in the final LLM verifier, about 380K in total.

determine the final choice among survivors. The Stage 1 EF verifier is typically a separately fine-tuned LLM that reads the trajectories produced by the policy [38]. Execution-free does not mean token-free: each candidate is processed by the verifier model again, adding verifier input and output tokens. This cost is particularly high on SWE tasks, where a trajectory commonly contains tens of thousands of tokens from reasoning, repository exploration, tool feedback, and code edits. Reprocessing such trajectories adds substantial token consumption to already expensive long-context inference [18, 20, 48]. For a recent open policy such as DeepSWE-Preview with $K = 16$ candidates, Stage 1 alone consumes approximately 850K tokens per task instance and the full verification workflow approximately 1,050K, exceeding the approximately 850K tokens used to generate the candidates. Candidate verification can thus rival generation as a source of inference-time token cost.

We introduce a token-efficient verification method called **LatentGate**. It filters candidate trajectories using hidden states collected during the generating process, introducing no additional LLM token cost and requiring no test execution. We represent each trajectory through three channels: reasoning, environment observations, and function calls. For each channel, we compare candidate states with positive and negative banks collected during policy training, aggregate the step-level distance margins, and rank candidates for the task instance. We take the minimum of the three channel ranks as the distance score, so a candidate scores highly only if it ranks highly in every channel. We also train a linear scorer on policy hidden states to assign each candidate a learned trajectory score. We rank these scores within each task instance, normalize the ranks, and average them with the candidates' distance-based scores to obtain the final gate scores. This learned signal forms part of the latent gate rather than a separate downstream verification stage. We use LatentGate to replace the highly token-costing Stage 1 of the hybrid workflow, retaining half the candidates so execution-based stages and the final LLM verifier operate on a smaller pool, significantly reducing the token cost of the verification workflow.

Our main contributions focus on:

- We introduce LatentGate, a token-free, execution-free filtering mechanism for long-horizon software engineering agents, and integrate it into an inference-time hybrid verification workflow.
- We design a three-channel trajectory representation that separately pools reasoning, observation, and function-call states already computed by the generating policy. We then develop a trajectory scoring method, which combines contrastive distances to successful and unsuccessful experience across the three channels through minimum-rank fusion, then integrates a learned trajectory score to rank candidates within each issue.
- We evaluate LatentGate across multiple software engineering agents and policies on SWE-bench Verified, demonstrating substantial verification-token savings while maintaining

competitive selection quality. On DeepSWE-Preview, it reduces total verification tokens by 62.1% while improving hybrid Best@16.

2 Related Work

Test-time scaling and agent search. Test-time compute can be spent on repeated sampling, adaptive stopping, parallel probing, search, and verifier-guided selection [2, 24, 39, 44, 45, 56]; on learned allocation and multi-agent pipelines [29, 42, 57, 59]; or on interaction, revision, and search, as in ReAct, Reflexion, Tree of Thoughts, and language-agent tree search [37, 51, 52, 58]. For SWE-bench, CodeMonkeys separates candidate coverage from selection and pays for selection with generated tests and a model-based selector [9]. We study the marginal cost of that selection when the policy already produces long trajectories.

Software agents and patch verification. SWE-bench evaluates generated patches against repository tests [16], and existing systems combine tool-using agents, search, execution signals, and learned execution-free verifiers [3, 15, 38, 47, 50]. SWE-Gym trains agents and verifiers on executable environments and reports Best@K under verifier-guided scaling [32], the recipe inherited by the hybrid cascades we build on, whereas SWE-Search spends verifier tokens on LLM value estimates at every search node [3]. Process supervision, code-agent reward models, and execution-free patch reasoning further strengthen candidate assessment [8, 14, 49]; we target the token cost of applying such assessment at inference time.

Verifiers and process rewards. Outcome verifiers and process supervision were developed for mathematical reasoning [7, 25, 27, 41, 43] and extended to Q-value ranking, practical process-reward design, and step-wise promise or progress for agents [6, 23, 46, 54]. Preference optimization offers a related route to comparative judgments [36], and RewardBench, reward-overoptimization scaling laws, and LLM-as-a-judge studies document the robustness limits of learned evaluators [10, 21, 55]. Our verifier pairs a non-parametric distance judgment with a lightweight learned score, both computed from cached policy states.

Signals in internal representations. Activations encode truth, confidence, hallucination, and reasoning correctness [4, 5, 11, 17, 19, 22, 28, 31, 53], and distance to expert representations can steer visuomotor policies toward familiar behavior [40]. We turn such evidence into candidate selection for long-horizon language agents: success and failure banks built during policy training supply outcome-sensitive experience, the three-channel decomposition preserves the structure of agent interaction, and online state collection avoids replaying a trajectory through another language model.

3 Method

3.1 Problem Setup

For each task instance x_m, a fixed policy generates K trajectories $\tau_{m,1}, \ldots, \tau_{m,K}$, each ending in a patch whose outcome $y_{m,i} \in \{0, 1\}$ is determined by the evaluation protocol. A selector s scores the candidates and returns $i^\star_m = \arg\max_{i \le K} s(x_m, \tau_{m,i})$; outcome labels and hidden evaluation tests are unavailable to it, and instance-supplied tests are used only by execution-based stages. Over M task instances we compare the selected resolution rate $\frac{1}{M}\sum_m y_{m,i^\star_m}$ with the oracle rate $\frac{1}{M}\sum_m \max_i y_{m,i}$ of the same pools; their gap counts successful candidates the selector missed. Our constraint is the incremental language-model tokens spent after generation, $V(s) = N_{\text{in}} + N_{\text{out}}$, which for a full-trajectory LLM verifier scales with the combined length of the candidates. A selector is *token-free* when $V(s) = 0$: it may reuse hidden states from the policy's ordinary forward passes but never replays a trajectory through another language model. It is *execution-free* when it launches no environment interaction or repository tests.

3.2 Overview

Figure 2 summarizes LatentGate. Outcome-labeled trajectories from policy training populate a positive and a negative bank of reasoning, observation, and function-call states. At inference, the same

states are collected from each candidate as it is generated, compared with the banks channel by channel, and reduced to a within-instance distance rank; a linear scorer trained on the same states supplies a second rank. LatentGate selects the top half of the pool without any additional LLM pass or test execution. The upper-right panel previews the resulting cost at $K = 16$ on DeepSWE-Preview: verification tokens fall from about 1035K to 392K per task instance (62.1%), including test generation.

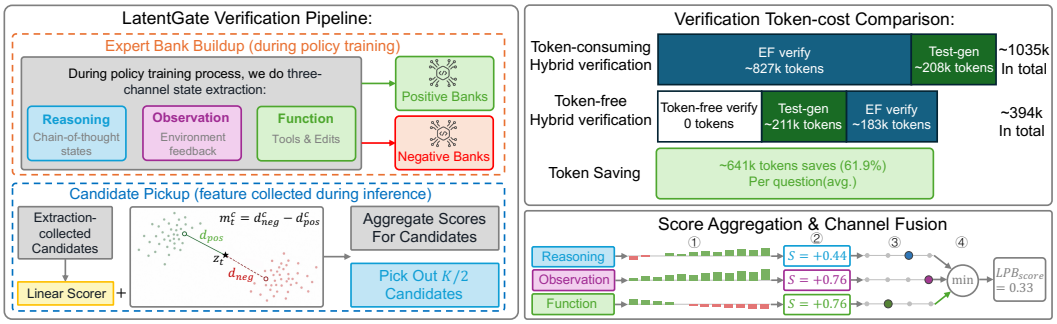

Figure 2: **LatentGate verification design. Left:** Reasoning, observation, and function states from outcome-labeled training trajectories form positive and negative banks. Inference reuses candidate states from ordinary policy forward passes, and Stage 1 averages the distance score with the linear scorer's within-instance rank to retain the top half of candidates. **Lower right:** Distance scoring: (1) retrieve the nearest positive and negative states and compute $m_t^c = d_{\text{neg}}^c - d_{\text{pos}}^c$; (2) aggregate by step-weighted mean; (3) rank within each task instance and channel; (4) take the minimum channel rank. **Upper right:** Average verification tokens per task instance for the original and token-free cascades at $K = 16$ (Section 4.2.1).

3.3 Three-channel state extraction

We segment each trajectory at function-call boundaries, which remain explicit even without reasoning tags, into three span types: **reasoning (CoT)**, assistant text outside function calls; **observation (Obs)**, environment responses such as retrieved code, command output, and test results; and **function (Fn)**, tool names and arguments, including code edits. Let $h_{i,p}^{(L)} \in \mathbb{R}^d$ denote the final-layer hidden state at token position p while the policy processes trajectory τ_i. At step t we mean-pool the states of each channel c:

$$z_{i,t}^c = \frac{1}{|\mathcal{S}_{i,t}^c|} \sum_{p \in \mathcal{S}_{i,t}^c} h_{i,p}^{(L)}, \qquad c \in \{\text{cot}, \text{obs}, \text{fn}\}, \tag{1}$$

where $\mathcal{S}_{i,t}^c$ is the span's token positions; the task description and structural delimiters belong to no channel. Pooling over whole spans keeps signals spread across long reasoning and code sequences. Bank and candidate states come from the same policy and are read from its ordinary forward passes, so extraction adds no model pass.

3.4 Positive and negative trajectory banks

From policy-training rollouts with executable outcome labels we form

$$\mathcal{D} = \{(x_j, \tau_j, \{z_{j,t}^c\}_{c,t}, y_j)\}_{j=1}^N, \qquad c \in \{\text{cot}, \text{obs}, \text{fn}\}, \tag{2}$$

where $y_j \in \{0, 1\}$ records whether the final patch resolves the task instance, and split $\mathcal{D}$ by outcome into a positive bank $\mathcal{B}_{\text{pos}}$ and a negative bank $\mathcal{B}_{\text{neg}}$. Each entry keeps all three channel sequences as fields of one trajectory record. Bank and evaluation repositories are disjoint, so the judgment must transfer beyond repository-specific content.

3.5 Distance scoring and latent filtering

For a candidate state $z_{i,t}^c$ we retrieve its nearest positive and negative bank states in the same channel, writing $\mathcal{Z}^c(\mathcal{B})$ for the channel-c states of bank $\mathcal{B}$:

$$d_{\text{pos}}^c(z) = \min_{u \in \mathcal{Z}^c(\mathcal{B}_{\text{pos}})} \|z - u\|_2, \qquad d_{\text{neg}}^c(z) = \min_{u \in \mathcal{Z}^c(\mathcal{B}_{\text{neg}})} \|z - u\|_2. \tag{3}$$

The signed margin is aggregated over steps with position weights,

$$m_{i,t}^c = d_{\text{neg}}^c(z_{i,t}^c) - d_{\text{pos}}^c(z_{i,t}^c), \qquad q_i^c = \frac{\sum_{t=0}^{T_i-1} t\, m_{i,t}^c}{\sum_{t=0}^{T_i-1} t}, \tag{4}$$

so that later steps, taken after environment feedback, count more; a positive value indicates proximity to successful rather than unsuccessful experience. We convert q_i^c to a normalized within-instance rank $r_i^c \in [0, 1]$, higher being better, and take

$$s_{\text{dist}}(\tau_i) = \min_{c \in \{\text{cot,obs,fn}\}} r_i^c, \tag{5}$$

so a candidate scores highly only if every channel ranks it highly. Appendix A.3 gives the rationale and cost accounting, and Section 4.3 ablates channel subsets, one-sided distances, and uniform step weights.

3.6 Linear scorer and signal fusion

A lightweight linear scorer inspired by SWIFT [13] complements the distance judgment. Trained on the same outcome-labeled trajectories, it reads pooled reasoning and function states from all transformer layers, excluding observations; a shared linear head maps each span to a gate logit and a local reward, and their gated average is the trajectory score, trained with binary cross-entropy against the outcome (Appendix A.5). At inference it reads only cached candidate states. Its score is converted to a within-instance rank $r_{i,\text{lin}} \in [0, 1]$ and fused with fixed equal weights,

$$s(\tau_i) = 0.5\, s_{\text{dist}}(\tau_i) + 0.5\, r_{i,\text{lin}}, \tag{6}$$

and the $\max(3, \lfloor K/2 \rfloor)$ highest-scoring candidates, the top half at $K = 16$, proceed to the execution-based stages and the final EF verifier. We use this fixed fusion in all main comparisons and ablations.

4 Experiments

4.1 Experimental setup

Task. We evaluate on the 500 human-validated GitHub issues in SWE-bench Verified [16]. Each trajectory terminates in a repository patch, whose correctness is determined by the official resolution evaluation. All selection methods are evaluated on the same candidate pools within each configuration, so performance differences reflect candidate selection rather than generation.

Policies, agents, and candidate pools. We study both cross-agent and cross-policy generalization. The cross-agent setting includes DeepSWE-Preview with DeepSWE Agent and R2EGym-32B with R2EGym Agent, the agentic scaffold released with that model. DeepSWE Agent denotes the official DeepSWE-Preview scaffold, comprising its prompt template, tool protocol, and agent configuration, and R2EGym Agent denotes the official R2E-Gym scaffold.[1] Both are implemented in the R2E-Gym framework and share its editing interface, but they are distinct agent configurations. The cross-policy setting fixes R2EGym Agent and compares R2EGym-32B and R2EGym-14B [15].

DeepSWE-Preview is RL-trained from Qwen3-32B [26], and R2EGym-32B is fine-tuned from Qwen2.5-Coder-32B-Instruct [15]. For each policy–agent configuration, a complete candidate pool contains 16 independently generated trajectories for each of the 500 task instances, totaling 8,000 trajectories. Verifier comparisons reuse the same pool, whereas changing the agent scaffold requires regenerating trajectories and the corresponding execution signals.

The trajectory banks are collected during policy training, following the DeepSWE-Preview training procedure [26]: for DeepSWE-Preview, we collected 1,383 successful and 1,383 unsuccessful trajectories from its training rollouts, on repositories disjoint from the evaluation repositories, and we apply the same procedure to every other policy, so each policy has a bank and a linear scorer in its own representation space. Evaluation outcomes are never used to construct either scoring signal.

[1] Official DeepSWE reproduction guide.

For each task instance and each K, we take 200 random draws of K candidates from the pool of 16 and report the mean Best@K over the draws. Candidate generation and the execution-based signals are not re-run. At $K = 16$ a draw is a random ordering of the whole pool, so the repetitions then differ only in how ties among equally scored candidates are broken; the EF baselines rank candidates by continuous stored probabilities and are therefore unaffected by tie-breaking.

Verifiers and controlled comparisons. At Stage 1, we compare DeepSWE-Verifier [1], R2EGym-Verifier [35], DEV matching-pairs [33], AgentPRM [46], and our LatentGate verifier. Each method retains $\max(3, \lfloor K/2 \rfloor)$ candidates, the top half at $K = 16$. Stages 2 and 3 use identical regression-test and generated-test signals within each configuration, with R2E-TestgenAgent providing test generation [34].

We use two controlled evaluation settings. In both, each EF baseline uses its own verifier in Stages 1 and 4, reusing its Stage 1 scores for final selection, whereas ours uses LatentGate in Stage 1 and the policy-associated EF verifier in Stage 4. The cross-agent experiment varies the policy and agent scaffold; the cross-policy experiment fixes R2EGym Agent and varies the policy size. Detailed checkpoint identities and implementation choices are provided in Appendix A.4.

Metrics. Our primary quality metric is Best@K, the percentage of task instances solved by the candidate selected from a pool of size K; hybrid Best@K is this rate after the complete four-stage cascade. Oracle Pass@K reports the corresponding candidate-generation ceiling. We measure verification cost after candidate generation, reporting EF-verifier tokens and total verification tokens including test generation; all token counts are measured from the calls actually made, EF scores are reused when available, and only calls actually executed count toward inference cost.

4.2 Main results

We evaluate hybrid verification across agent configurations and policy sizes. The cross-agent experiment compares four released EF verifiers with LatentGate, each baseline serving in both EF stages and ours paired with the policy-associated final verifier; the cross-policy experiment compares each policy's own verifier with LatentGate under a fixed scaffold.

4.2.1 Cross-agent comparison

We compare DeepSWE-Preview with DeepSWE Agent, and R2EGym-32B with R2EGym Agent; the first two share the R2E-Gym editing interface but are distinct policy–agent configurations.

Table 1: Cross-agent hybrid results on SWE-bench Verified at $K = 16$. Best@16 is the resolution rate (%); EF and Total are EF-verifier and total verification tokens per task instance, in thousands. Dashes mark results not yet available.

	DeepSWE Agent			R2EGym Agent		
Stage 1 verifier	Best@16	EF	Total	Best@16	EF	Total
DeepSWE-Verifier	59.26	826.7	1035.1	49.70	367.4	599.4
R2EGym-Verifier	57.95	774.1	982.0	46.56	366.2	599.6
DEV matching-pairs	58.75	826.7	1037.6	48.38	367.4	599.4
AgentPRM	59.76	810.9	1022.8	46.56	358.3	586.8
Ours	60.06	181.0	391.9	47.73	76.0	304.9

Table 1 reports hybrid Best@16 and verification cost for the five Stage-1 verifiers under the protocol of Section 4.1. On the DeepSWE pool no released verifier exceeds LatentGate (60.06%). On the R2EGym-32B pool, DeepSWE-Verifier reaches 49.70% and DEV matching-pairs 48.38%, above LatentGate's 47.73%, whereas AgentPRM never improves on the policy's own verifier (Table 7); this order follows the verifiers' within-instance discrimination reported in Section 4.2.3. Each released verifier scores all 16 candidates and costs 358–367K EF tokens per task instance against 76.0K for LatentGate, so a stronger LLM verifier buys about two points on this pool at roughly 4.8× the EF tokens and twice the total verification cost.

Figure 3 tracks both quantities as the candidate budget grows, with the pool oracle as the generation ceiling. On DeepSWE-Preview, ours and the original hybrid stay within half a point of each other up to $K = 8$ and ours is ahead from $K = 12$ on; on R2EGym-32B the original cascade is marginally ahead for K between 4 and 12, the curves cross near $K = 14$, and LatentGate leads only at the

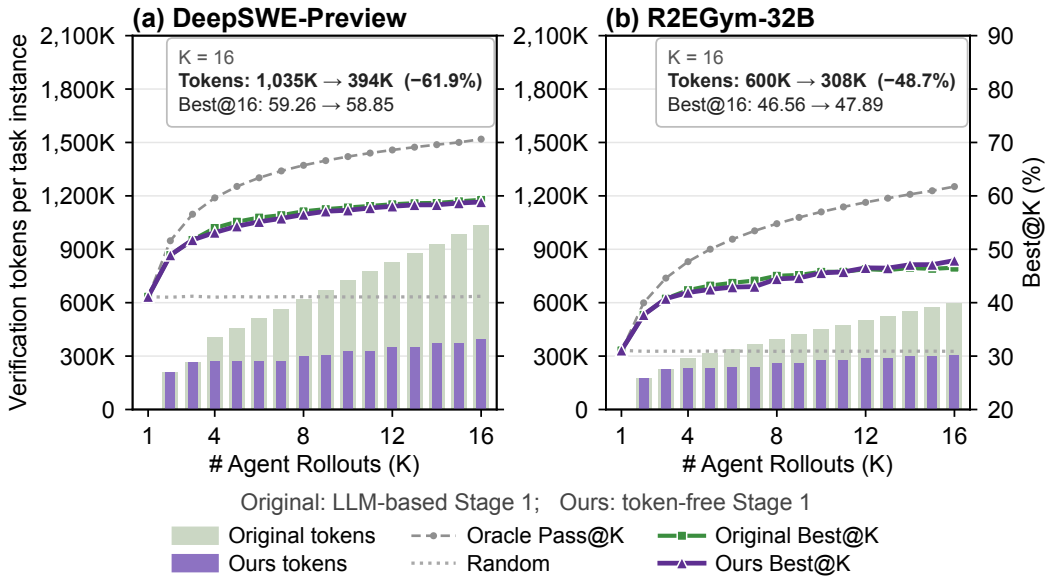

Figure 3: **Verification cost and selection quality as the candidate budget grows.** Bars: verification tokens per task instance, including test generation (left axis); curves: Best@K and Oracle Pass@K (right axis), with random selection for reference. The original hybrid uses the policy's verifier in Stages 1 and 4; ours replaces it in Stage 1 with LatentGate.

largest budgets. The token saving instead appears as soon as the gate removes candidates ($K \geq 4$) and widens with K, because the original cascade's verification tokens grow linearly with K while ours grow slowly.

4.2.2 Cross-policy comparison

We hold R2EGym Agent fixed and compare R2EGym-32B and R2EGym-14B. Each policy supplies its own candidate pool, its own bank and linear scorer, and its released verifier, R2EGym-Verifier, which the original cascade uses in Stages 1 and 4 and ours only in Stage 4. This comparison tests whether the token-free first-stage judgment remains effective across policy sizes without changing the agent interface.

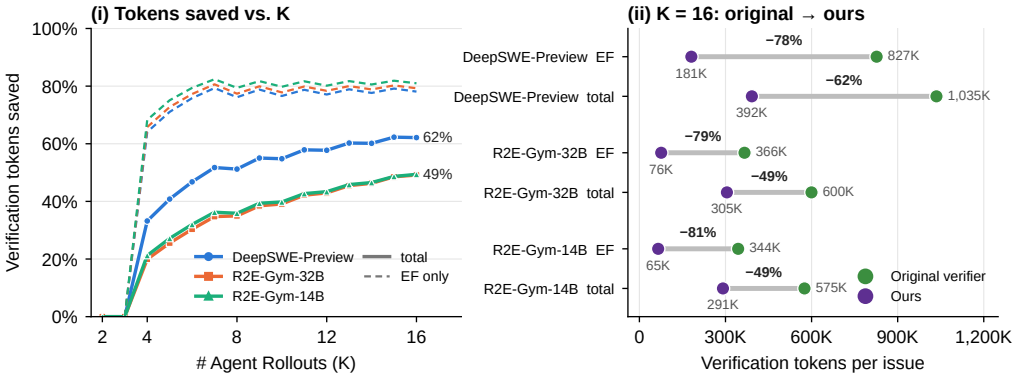

Figure 4: **Verification tokens saved across policies.** (i) Share of verification tokens saved as K grows (solid: total including test generation; dashed: EF-verifier tokens only). (ii) Tokens per task instance at $K = 16$ with each policy's own verifier and with LatentGate in Stage 1.

Figure 4 reports the verification tokens saved under the same accounting as the cross-agent comparison, with the DeepSWE-Preview configuration shown for reference. At $K = 16$, EF-verifier

tokens fall from 366K to 76K per task instance on R2EGym-32B (79%) and from 344K to 65K on R2EGym-14B (81%); total verification tokens, including test generation, fall from 600K to 305K and from 575K to 291K (49% for both). For the R2EGym policies the saved share is essentially flat from $K \approx 6$ onward (panel i), so the benefit does not depend on a particular candidate budget. Hybrid Best@16 (Table 2) rises from 46.56% to 47.73% on R2EGym-32B and falls from 43.28% to 42.95% on R2EGym-14B. The 14B decrease is very limited, 0.33 points or 0.8% of the original rate, and it comes with an 81% reduction in EF-verifier tokens and a 49% reduction in total verification tokens; Section 4.2.3 traces the small loss to lower Stage 1 retention than the LLM verifier on that pool.

4.2.3 Gate quality: oracle retention after Stage 1

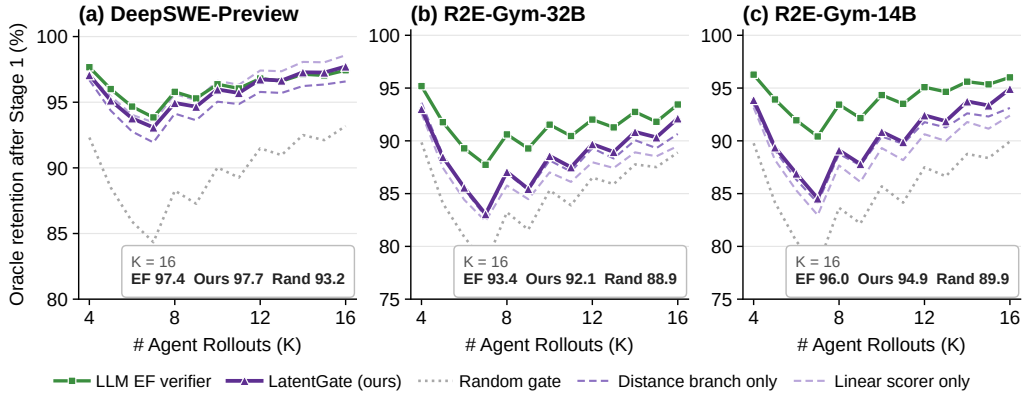

Figure 5: **Oracle retention after Stage 1.** Fraction of task instances that still hold a correct candidate after the Stage-1 cut, among task instances whose K candidates contain one. The cut keeps $\max(3, \lfloor K/2 \rfloor)$ candidates, so $K \leq 3$ is omitted and odd K retain proportionally fewer, which produces the sawtooth.

Best@16 after the full cascade credits the Stage 1 gate together with three downstream stages. To isolate the gate we measure its *oracle retention*: among task instances whose K candidates contain a correct patch, the fraction whose retained $\max(3, \lfloor K/2 \rfloor)$ candidates still contain one, the only quantity Stage 1 controls directly. Figure 5 plots it against K for the LLM EF verifier, LatentGate, its two components, and a random gate on the same 200 draws as Figure 3; Table 2 reports $K = 16$ with Best@16. As a candidate-level complement, Table 6 (Appendix A.7) reports each scorer's within-instance AUC against the hidden-test outcomes, used only for this post-hoc analysis: the LLM verifiers discriminate better within a task instance than the token-free gate on every pool (0.747–0.810 against 0.622–0.745), the gap that retention shows at small K.

Table 2: Stage-1 gate quality at $K = 16$ (%). Ret.: oracle retention after the gate, over task instances whose pool contains a correct candidate; Best@16: resolution rate after the full cascade.

	DeepSWE-Preview		R2EGym-32B		R2EGym-14B	
Stage-1 gate	Ret.	Best@16	Ret.	Best@16	Ret.	Best@16
LLM EF verifier	97.44	59.26	93.44	46.56	96.01	43.28
LatentGate (ours)	**97.72**	**60.06**	**92.13**	**47.73**	**94.93**	**42.95**
Random gate	93.19	55.98	88.90	45.48	89.94	40.98

Two observations follow. First, LatentGate retains far more than a random gate at every K: at $K = 16$ it keeps a correct candidate on 92–98% of solvable task instances across the three pools, against 89–93% for the random gate, without reading a token. Second, it matches the LLM verifier on DeepSWE-Preview (97.72% vs. 97.44%) and trails it by 1.1–1.3 points on the two R2EGym pools; on R2EGym-32B this gap does not surface in Best@16, which is higher with LatentGate (47.73% vs. 46.56%), because the execution stages and the Stage-4 EF make the final choice over the retained set. The dashed curves show that neither component alone is stronger than the fused gate on every pool, which is why we keep the equal-weight fusion.

4.3 Ablation study

We ablate three design choices of the distance branch at $K = 16$, with the policy, scaffold, banks, execution signals, linear scorer, and Stage 4 verifier fixed within each configuration: channel selection, distance construction, and temporal weighting. Channel subsets take the minimum rank over the selected channels and are fused with the linear scorer as in the full method. Tables 3–5 report hybrid Best@16; the last two also report Stage-1 retention.

4.3.1 Channel selection

Restricting the distance branch to single channels or pairs, with the learned signal and downstream stages fixed, lowers Best@16 in every case (Table 3). The best pair, observation + function, comes closest (59.86% and 47.10% against 60.06% and 47.73%), single channels lose more, and their relative value differs by agent: the function channel alone is comparatively strong on R2EGym Agent. The three channels therefore carry complementary information even when the learned linear signal is available.

Table 3: Channel selection ablation of LatentGate at $K = 16$: hybrid Best@16 (%) with the distance branch restricted to subsets of the reasoning (R), observation (O), and function (F) channels.

Distance channels	**R+O+F**	R+O	R+F	O+F	R	O	F
DeepSWE Agent	**60.06**	59.26	59.66	59.86	59.46	59.26	59.46
R2EGym Agent	**47.73**	46.60	46.90	47.10	46.71	46.46	47.25

4.3.2 Distance construction

The full method uses the contrastive margin $m^c_{i,t} = d^c_{\text{neg}} - d^c_{\text{pos}}$ of Equation 4. We compare it with the one-sided distances d^c_{pos} (proximity to successful trajectories, smaller is better) and d^c_{neg} (separation from unsuccessful trajectories, larger is better), everything else unchanged.

The margin is best on both agents (Table 4), and the two sides contribute asymmetrically in the same direction: d^c_{neg} alone loses 0.20 and 0.43 points, d^c_{pos} alone 2.01 and 3.57, so separation from unsuccessful experience carries most of the signal. The asymmetry is larger before fusion, where the one-sided variants lose 1.37 and 5.03 points on DeepSWE Agent and 0.02 and 3.17 on R2EGym Agent, so the learned scorer recovers part of what a one-sided distance drops. Stage-1 retention orders the three constructions the same way, so the margin also hands the execution stages a retained half that holds a correct candidate more often.

Table 4: Distance construction ablation at $K = 16$ (%).

Distance score	Best@16	Ret.
DeepSWE Agent		
$d^c_{\text{neg}} - d^c_{\text{pos}}$	**60.06**	**97.72**
d^c_{pos}	58.05	97.30
d^c_{neg}	59.86	97.44
R2EGym Agent		
$d^c_{\text{neg}} - d^c_{\text{pos}}$	**47.73**	**92.13**
d^c_{pos}	44.16	89.31
d^c_{neg}	47.30	91.48

Table 5: Step-weighting ablation at $K = 16$ (%).

Aggregation	Best@16	Ret.
DeepSWE Agent		
Step-weighted mean	**60.06**	**97.72**
Uniform step mean	59.66	97.44
R2EGym Agent		
Step-weighted mean	**47.73**	**92.13**
Uniform step mean	46.36	90.16

4.3.3 Step weighting

Finally, we replace the step-weighted aggregation of the per-step margins with the uniform mean $q^c_{i,\text{uniform}} = \frac{1}{T_i} \sum_{t=0}^{T_i-1} m^c_{i,t}$, everything else unchanged. The uniform mean is worse on both agents: Best@16 falls by 0.40 and 1.37 points and Stage-1 retention by 0.28 and 1.97 points (Table 5), at every candidate budget and whether the distance branch is fused with the learned scorer or used alone (-0.50 and -1.32). States reached after execution feedback are more informative than the trajectory average.

4.4 Limitations

Our method requires access to internal activations and therefore may not directly apply when the policy is available only through a hosted API. The learned scorer may also depend on the policy checkpoint and agent scaffold on which it was trained. Execution-derived labels reflect the available test suite and can fail to capture defects that are not exercised by those tests.

Representation quality may additionally degrade for long trajectories because pooled features can dilute localized signals, while a learned scorer may exploit superficial correlates such as trajectory length or tool-usage patterns. Distribution shift across repositories, task types, agent scaffolds, or policy checkpoints may therefore reduce selection quality even when the scorer achieves high in-domain accuracy. Finally, the proposed selection procedure can only rank candidates that are present in the generated pool and cannot recover a correct patch if no such candidate was produced.

5 Conclusion

We presented a framework for token-free verification of software engineering agents using reasoning, observation, and function-call channels. Contrastive success/failure judgments and within-instance aggregation turn these signals into a candidate score without replaying the trajectory through another language model. Across DeepSWE-Preview and R2EGym-32B, LatentGate reduces EF-verifier tokens by 78.1% and 79.2% at $K = 16$ while improving hybrid Best@16 from 59.26% to 60.06% and from 46.56% to 47.73%. Integrated into the complete hybrid cascade, it reduces total verification cost by 62.1% and 49.1%, respectively. The consistency of the EF-token savings across policies with different trajectory lengths supports structured policy-state reuse as a practical replacement for first-stage LLM verification.

AI use statement

Generative AI assisted with organizing the manuscript and drafting prose and mathematical formulations from the authors' research notes. The authors reviewed the resulting technical claims and remain responsible for the manuscript, experiments, and references.

Reproducibility statement

Reproducibility materials will include bank construction procedures, split identifiers, label provenance, extraction code, scorer configurations, and evaluation scripts. Appendix A lists the corresponding reporting details.

References

[1] Agentica. DeepSWE-Verifier. Hugging Face model repository, 2025. URL `https://huggingface.co/agentica-org/DeepSWE-Verifier`.

[2] Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12375–12396. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.761. URL `https://aclanthology.org/2023.emnlp-main.761/`.

[3] Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, pages 64485–64515, 2025. URL `https://proceedings.iclr.cc/paper_files/paper/2025/file/a1e6783e4d739196cad3336f12d402bf-Paper-Conference.pdf`.

[4] Amos Azaria and Tom Mitchell. The internal state of an LLM knows when it's lying. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 967–976, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.68. URL `https://aclanthology.org/2023.findings-emnlp.68/`.

[5] Collin Burns, Haotian Ye, Dan Klein, and Jacob Steinhardt. Discovering latent knowledge in language models without supervision. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ETKGuby0hcs.

[6] Sanjiban Choudhury. Process reward models for llm agents: Practical framework and directions, 2025. URL https://arxiv.org/abs/2502.10325.

[7] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

[8] Mahir Labib Dihan and Md Ashrafur Rahman Khan. Swe-shepherd: Advancing prms for reinforcing code agents, 2026. URL https://arxiv.org/abs/2604.10493.

[9] Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. Codemonkeys: Scaling test-time compute for software engineering, 2025. URL https://arxiv.org/abs/2501.14723.

[10] Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization, 2022. URL https://arxiv.org/abs/2210.10760.

[11] Amirhosein Ghasemabadi and Di Niu. Can llms predict their own failures? self-awareness via internal circuits, 2026. URL https://arxiv.org/abs/2512.20578.

[12] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://doi.org/10.1038/s41586-025-09422-z.

[13] Jizhou Guo, Zhaomin Wu, Hanchen Yang, and Philip S. Yu. Mining intrinsic rewards from LLM hidden states for efficient best-of-N sampling. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2026. doi: 10.1145/3770854.3780302. URL https://arxiv.org/abs/2505.12225.

[14] Hao Han, Jin Xie, Xuehao Ma, Weiquan Zhu, Ziyao Zhang, ZhiLiang Long, Hongkai Chen, and Qingwen Ye. Swe-trace: Optimizing long-horizon swe agents through rubric process reward models and heuristic test-time scaling, 2026. URL https://arxiv.org/abs/2604.14820.

[15] Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural Environment Generation and Hybrid Verifiers for Scaling Open-Weights SWE Agents. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=7evvwwdo3z.

[16] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf.

[17] Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know, 2022. URL https://arxiv.org/abs/2207.05221.

[18] Jiin Kim, Byeongjun Shin, Jinha Chung, and Minsoo Rhu. The cost of dynamic reasoning: Demystifying AI agents and test-time scaling from an AI infrastructure perspective. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–16. IEEE Computer Society, 2026. doi: 10.1109/HPCA68181.2026.11408569. URL https://doi.org/10.1109/HPCA68181.2026.11408569.

[19] Jannik Kossen, Jiatong Han, Muhammed Razzak, Lisa Schut, Shreshth Malik, and Yarin Gal. Semantic entropy probes: Robust and cheap hallucination detection in llms, 2024. URL https://arxiv.org/abs/2406.15927.

[20] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

[21] Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. RewardBench: Evaluating Reward Models for Language Modeling. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1755–1797. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.96. URL https://aclanthology.org/2025.findings-naacl.96/.

[22] Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 41451–41530. Curran Associates, Inc., 2023. doi: 10.52202/075280-1797. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/81b8390039b7302c909cb769f8b6cd93-Paper-Conference.pdf.

[23] Wendi Li and Yixuan Li. Process reward model with q-value rankings. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, pages 14708–14726, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/26494b66ae1114d314673e25b4967288-Paper-Conference.pdf.

[24] Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 14751–14768, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/3fe2a777282299ecb4f9e7ebb531f0ab-Paper-Conference.pdf.

[25] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

[26] Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. DeepSWE: Training a Fully Open-sourced, State-of-the-Art Coding Agent by Scaling RL. Together AI Research Blog, 2025. URL https://www.together.ai/blog/deepswe.

[27] Qianli Ma, Haotian Zhou, Tingkai Liu, Jianbo Yuan, Pengfei Liu, Yang You, and Hongxia Yang. Let's reward step by step: Step-level reward model as the navigators for reasoning, 2023. URL https://arxiv.org/abs/2310.10080.

[28] Samuel Marks and Max Tegmark. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=aajyHYjjsk.

[29] Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Philip Torr, Ivan Laptev, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. MALT: Improving reasoning with multi-agent LLM training. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=jXP9bgFack.

[30] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20275–20321. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

[31] Hadas Orgad, Michael Toker, Zorik Gekhman, Roi Reichart, Idan Szpektor, Hadas Kotek, and Yonatan Belinkov. Llms know more than they show: On the intrinsic representation of llm hallucinations. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, pages 66880–66913, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a712d461e57201efe35d429a6f1731c1-Paper-Conference.pdf.

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

[33] R2E-Edits. DeepSWE Verifier (matching-pairs-v1). Hugging Face model repository, 2025. URL https://huggingface.co/r2e-edits/deepswe-swebv-eval-n16-verifier-qwen3-lora64-matching-pairs-v1.

[34] R2E-Gym. R2E-TestgenAgent. Hugging Face model repository, 2025. URL https://huggingface.co/R2E-Gym/R2E-TestgenAgent.

[35] R2E-Gym. R2EGym-Verifier. Hugging Face model repository, 2025. URL https://huggingface.co/R2E-Gym/R2EGym-Verifier.

[36] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741. Curran Associates, Inc., 2023. doi: 10.52202/075280-2338. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a85b405ed65c6477a4fe8302b5e06ce7-Paper-Conference.pdf.

[37] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

[38] KaShun Shum, Binyuan Hui, Jiawei Chen, Lei Zhang, X. W., Jiaxi Yang, Yuzhen Huang, Junyang Lin, and Junxian He. SWE-RM: Execution-free feedback for software engineering agents. In *International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10010427.

[39] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, pages 10131–10165, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1b623663fd9b874366f3ce019fdfdd44-Paper-Conference.pdf.

[40] Zhanyi Sun and Shuran Song. Latent policy barrier: Learning robust visuomotor policies by staying in-distribution. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 174280–174305. Curran Associates, Inc., 2025. doi: 10.52202/085713-5797. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/feaf23597f3c93803d1b4230aadc8f44-Paper-Conference.pdf.

[41] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

[42] Fali Wang, Hui Liu, Zhenwei Dai, Jingying Zeng, Zhiwei Zhang, Zongyu Wu, Chen Luo, Zhen Li, Xianfeng Tang, Qi He, and Suhang Wang. Agenttts: Large language model agent for test-time compute-optimal scaling strategy in complex tasks. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 98396–98433. Curran Associates, Inc., 2025. doi: 10.52202/085713-3288. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/8d9bbba8cac9cabb54e85ee7f21441c8-Paper-Conference.pdf.

[43] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

[44] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

[45] Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for llm problem-solving. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors,

International Conference on Learning Representations, volume 2025, pages 55824–55845, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/8c3caae2f725c8e2a55ecd600563d172-Paper-Conference.pdf.

[46] Zhiheng Xi, Chenyang Liao, Guanyu Li, Zhihao Zhang, Wenxiang Chen, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. AgentPRM: Process reward models for LLM agents via step-wise promise and progress. In *Proceedings of the ACM Web Conference 2026*, pages 4184–4195. Association for Computing Machinery, 2026. doi: 10.1145/3774904.3792551. URL https://doi.org/10.1145/3774904.3792551.

[47] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE): 801–824, 2025. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754.

[48] Wenya Xie, Shaochen, Zhong, Hoang Anh Duy Le, Zhaozhuo Xu, Jianwen Xie, and Zirui Liu. Word salad chopper: Reasoning models waste a ton of decoding budget on useless repetitions, self-knowingly, 2025. URL https://arxiv.org/abs/2511.00536.

[49] Junjielong Xu, Boyin Tan, Xiaoyuan Liu, Chao Peng, Pengfei Gao, and Pinjia He. Scalable supervision for software agents via patch reasoning, 2026. URL https://arxiv.org/abs/2510.22775. Accepted to Findings of EMNLP 2026.

[50] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

[51] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, pages 11809–11822, 2023. doi: 10.52202/075280-0517. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

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

[53] Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. Reasoning models know when they're right: Probing hidden states for self-verification. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=O6I0Av7683.

[54] Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning, 2025. URL https://arxiv.org/abs/2501.07301.

[55] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc., 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

[56] Tong Zheng, Chengsong Huang, Runpeng Dai, Yun He, Rui Liu, Xin Ni, Huiwen Bao, Kaishen Wang, Hongtu Zhu, Jiaxin Huang, Furong Huang, and Heng Huang. Parallel-probe: Towards efficient parallel thinking via 2d probing, 2026. URL https://arxiv.org/abs/2602.03845.

[57] Tong Zheng, Haolin Liu, Chengsong Huang, Huiwen Bao, Sheng Zhang, Rui Liu, Runpeng Dai, Ruibo Chen, Chenxi Liu, Tianyi Xiong, Xidong Wu, Hongming Zhang, and Heng Huang. Llms improving llms: Agentic discovery for test-time scaling, 2026. URL https://arxiv.org/abs/2605.08083.

[58] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 62138–62160. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

[59] King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. Scaling test-time compute for llm agents, 2025. URL https://arxiv.org/abs/2506.12928.

A Implementation and Reporting Details

A.1 Judgment bank

Record task-instance and repository identifiers, collection dates, policy and expert checkpoints, sampling settings, outcome definitions, unsuccessful environment runs, deduplication, and split rules. Define whether the reported trajectory length counts unique transcript tokens or repeated inputs across policy calls.

A.2 State extraction

Record the final-layer extraction point, exact token positions, treatment of tool outputs, extraction timing, pooling, and feature precision for the main distance-based method.

A.3 Distance aggregation, selection, and cost accounting

Contrastive distance. The margin in Equation 4 compares a query state with successful and unsuccessful experience. A positive value means the nearest successful state is closer. A one-sided distance supplies only proximity to one outcome; contrasting both outcomes provides relative evidence. The channel-specific retrieval uses fields from the same two trajectory banks, not separately constructed channel banks.

Temporal and channel aggregation. Step-position weights emphasize evidence later in the interaction, after the agent has received environment feedback and revised its solution. The unweighted variant is evaluated in Section 4.3. Channel scores can have different scales and distributions, so raw addition can favor a channel because of its magnitude. Within-instance ranks compare candidates on a common scale without requiring calibrated success probabilities. Minimum-rank fusion limits the distance score whenever any channel ranks a candidate poorly; it does not impose a separate rejection threshold. For example, ranks of 0.67, 1.00, and 0.33 produce $s_{\text{dist}} = 0.33$. The learned signal is incorporated afterwards through Equation 6.

Candidate selection. As a standalone selector, LatentGate returns the patch with the highest fused score. In the hybrid, it retains half the pool before execution-based filtering and final LLM verification. Candidate features are collected during policy inference, whereas outcome-labeled bank features are collected during policy training. No trajectory is replayed through an additional LLM to compute the latent gate.

Cost accounting. The gate's computation comprises state extraction, pooling, retrieval, and scoring:

$$C_{\text{select}} = C_{\text{extract}} + C_{\text{pool}} + C_{\text{score}}. \tag{7}$$

Token-free refers to additional LLM input and output tokens, not zero computation. A text verifier that rereads K trajectories of average length $\bar{n}$ processes approximately $K\bar{n}$ input tokens before prompt overhead. This is a token-count example, not a latency or FLOP estimate. Token accounting distinguishes EF calls from test-generation costs, records cached-input treatment, and excludes candidate generation. Systems reporting should separately measure state-collection overhead, retrieval/scoring time, peak memory, and policy throughput. Bank construction and scorer training are offline costs.

A.4 EF verifier configurations

The policy-associated official checkpoints are `agentica-org/DeepSWE-Verifier`, a Qwen3-14B LoRA adapter, and `R2E-Gym/R2EGym-Verifier`, a Qwen2.5-Coder-14B-Instruct LoRA adapter. Both released adapter configurations specify rank 64 and $\alpha = 128$ [1, 35]. The official hybrid aggregation selects by stored EF probabilities after regression and reproduction-test filtering; it does not load a distinct final-stage verifier.[2] In the cross-agent experiment, each EF baseline uses its own checkpoint in both EF stages; ours uses the policy-associated official checkpoint only in Stage 4.

[2] Official hybrid aggregation implementation.

The cross-policy experiment follows the same assignment: each baseline uses its own checkpoint in both EF stages, and ours uses R2EGym-Verifier in Stage 4 for every R2EGym policy.

DeepSWE-Verifier and DEV matching-pairs are Qwen3-14B LoRA adapters with rank 64 and $\alpha = 128$. Their published model cards list a learning rate of 10^{-5}, two training epochs, seed 42, and cosine learning-rate decay with a warmup ratio of 0.05 [1, 33]. The released trainer states record final global steps of 7,532 and 5,532, respectively. The DEV matching-pairs model card identifies its training dataset as `deepswe-verifier-only-matching-pairs-v1`.

A.5 Learned scorer architecture and training

The learned scorer consumes cached, step-pooled hidden states rather than trajectory text. For every step, we retain the model-authored reasoning and function spans and exclude environment observations, matching SWIFT's assistant-only scoring scope. The two span types are interleaved in trajectory order. Let $u_{i,j}^{(\ell)} \in \mathbb{R}^d$ denote pooled span j from layer ℓ. We concatenate the representations from all L transformer layers without normalization:

$$\tilde{u}_{i,j} = [u_{i,j}^{(1)}; u_{i,j}^{(2)}; \ldots; u_{i,j}^{(L)}] \in \mathbb{R}^{Ld}. \tag{8}$$

A shared linear head produces a gate logit and local reward,

$$(\tilde{g}_{i,j}, v_{i,j}) = W\tilde{u}_{i,j} + b, \qquad W \in \mathbb{R}^{2\times Ld}, \quad b \in \mathbb{R}^2, \tag{9}$$

and the trajectory logit is their gated average:

$$q_{i,\mathrm{lin}} = \frac{\sum_j \sigma(\tilde{g}_{i,j}) v_{i,j}}{\max\left(\sum_j \sigma(\tilde{g}_{i,j}), 10^{-8}\right)}. \tag{10}$$

The implementation realizes W as one bias-free d-to-2 projection per layer plus a shared two-dimensional bias. This is algebraically equivalent to one Ld-to-2 linear layer while avoiding explicit materialization of the concatenated tensor, and it contains $2Ld + 2$ trainable parameters.

We assign each trajectory the binary target $y_i = \mathbb{1}[\mathrm{reward}_i \geq 0.5]$ and minimize binary cross-entropy with $q_{i,\mathrm{lin}}$ as the logit. The bank is split into 80% training and 20% validation trajectories with random seed 0. We optimize with AdamW using a learning rate of 3×10^{-6}, weight decay of 10^{-5}, gradient accumulation over 16 trajectories, and at most 40 epochs, retaining the checkpoint with the lowest validation loss. Query trajectories are scored once with the selected checkpoint. For fusion, these logits are converted to normalized ranks within the candidate set for each task instance before applying Equation 6.

A.6 R2EGym token accounting

EF-verifier tokens are computed from the prompt length of every verifier call actually made, not from a global mean. The average EF prompt contains 22.9K tokens over the full R2EGym-32B candidate pool and 18.9K tokens among the $K = 16$ candidates that survive to the final verifier; the corresponding DeepSWE-Preview means are 51.7K and 46.2K tokens. Most of the saving therefore comes from the smaller number of EF calls, with a smaller contribution from the shorter average length of the surviving trajectories.

Stage 3 tests are generated for all 500 task instances of every configuration with the open-source DeepSWE-Preview execution-based verifier, following the official DeepSWE-Preview procedure, and their token cost is measured from the generation logs: a Stage 3 invocation costs 244.3K tokens on average. For R2EGym-32B, Stage 3 runs on 95.5% of task instances in the original cascade and on 93.7% in ours, so the totals decompose as $366.2\mathrm{K} + 0.955 \times 244.3\mathrm{K} \approx 600\mathrm{K}$ and $76.0\mathrm{K} + 0.937 \times 244.3\mathrm{K} \approx 305\mathrm{K}$ tokens per task instance, the 49.1% saving reported in Table 1 and Figure 4; R2EGym-14B decomposes in the same way with its own invocation rates (575K to 291K).

A.7 Verifier discrimination and smaller candidate budgets

Table 6 gives the within-instance AUC defined in Section 4.2.3 for every Stage-1 scorer on the three candidate pools. Pooling all candidates of a pool instead of averaging within task instances gives

0.832, 0.787, 0.803, and 0.770 on DeepSWE-Preview, 0.818, 0.796, 0.800, and 0.649 on R2EGym-32B, and 0.830, 0.808, 0.800, and 0.641 on R2EGym-14B for DeepSWE-Verifier, R2EGym-Verifier, DEV matching-pairs, and AgentPRM, the same ordering on the R2EGym pools; the rank-valued gate has no meaningful pooled AUC, since its scores are relative to one task instance. Table 7 reports the cross-agent hybrid Best@K of Table 1 at $K = 4$, 8, and 12, the values behind the smaller-budget statements of Section 4.2.1.

Table 6: Within-instance discrimination of every Stage-1 scorer. Each cell is the AUC of the scorer's candidate score against the candidates' hidden-test outcomes, computed within one task instance and averaged over the task instances whose rollouts contain both a resolved and an unresolved candidate (309, 283, and 260 task instances for the three pools). The outcomes are used only for this post-hoc analysis and never enter candidate selection. A random gate scores 0.500.

Stage-1 scorer	DeepSWE-Preview	R2EGym-32B	R2EGym-14B
DeepSWE-Verifier	0.810	0.773	0.793
R2EGym-Verifier	0.785	0.747	0.773
DEV matching-pairs	0.812	0.765	0.787
AgentPRM	0.788	0.654	0.660
LatentGate (ours)	0.745	0.641	0.622
distance only	0.698	0.598	0.592
linear only	0.758	0.650	0.616

Table 7: Hybrid Best@K (%) of the cross-agent comparison at smaller candidate budgets, under the protocol of Table 1: each released verifier occupies Stages 1 and 4, and LatentGate prunes at Stage 1 with the policy's verifier at Stage 4. The $K = 16$ column repeats the main table.

	DeepSWE-Preview				R2EGym-32B			
Stage-1 verifier	K=4	K=8	K=12	K=16	K=4	K=8	K=12	K=16
DeepSWE-Verifier	53.99	57.10	58.43	59.26	42.65	46.11	48.23	49.70
R2EGym-Verifier	53.67	56.38	57.59	57.95	42.35	45.04	46.27	46.56
DEV matching-pairs	53.71	56.70	57.82	58.75	42.49	45.58	47.38	48.38
AgentPRM	53.55	56.70	58.40	59.76	41.47	44.22	45.74	46.56
LatentGate (ours)	53.45	56.92	58.64	60.06	41.79	44.23	46.12	47.73

A.8 Evaluation and resources

Record prompts, context truncation, candidate seeds, tie breaking, test availability, inference engine, hardware, concurrency, timing boundaries, memory measurement, and all costs of bank construction. Keep hidden evaluation outcomes separate from every input used for candidate selection.