TEST-TIME SCALING VIA ERROR LOCALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling inference-time computation has emerged as a reliable method to improve the performance of large language models on complex reasoning and programming tasks. However, standard approaches such as independent sampling and sequential multi-turn refinement operate without token-level credit assignment, resulting in computational inefficiency, since valid reasoning prefixes are frequently discarded. In this work, we introduce Test-Time Scaling via Error Localization (TTEL), an inference-time algorithm that utilizes fixed or environment feedback to perform token-level error localization. By comparing conditional probabilities under informed feedback against a non-diagnostic baseline, TTEL isolates the step at which an error occurred. The algorithm then truncates the trajectory and branches a new generation, maximally reusing the valid prefix. Extensive evaluations demonstrate that TTEL establishes strictly dominating Pareto frontiers across sequential reasoning domains, measured by pass-at-k vs. generated-token cost. With Qwen3-8B on LiveCodeBench, TTEL attains a pass@64 of 71.0% while generating approximately half as many tokens as independent sampling (360.4k vs. 735.0k). Generalizing to math benchmarks AIME-2025 and HMMT-2025, TTEL cleanly outperforms competing test-time baselines across both Qwen3-8B and Qwen3-4B-Thinking-2507.

1 INTRODUCTION

Scaling inference-time computation improves the performance of LLMs on complex reasoning and programming tasks (Snell et al., 2024; DeepSeek-AI, 2025). The main approach for this is to generate multiple independent samples and select the best candidate via majority voting (Wang et al., 2023) or verification against test cases (Jain et al., 2024). While such best-of-K strategies reliably improve accuracy as K increases, they scale inefficiently: each sample is generated without knowledge of prior attempts, leading to redundant exploration of similar trajectories. For problems that require long reasoning chains, where a single early error invalidates the entire generation, this redundancy is wasteful.

Sequential multi-turn generation attempts to address this by incorporating environmental feedback into the context window (Madaan et al., 2023). The model generates a subsequent attempt conditioned on the failure of its previous output. However, sequential refinement often stagnates: when faced with genuinely difficult reasoning challenges, models frequently repeat similar logical errors across turns or fail to identify the precise point where the logic diverged (Kumar et al., 2024). The context window fills with unsuccessful attempts without yielding a correct solution.

A deeper issue is that post-hoc feedback is inherently *trajectory-conditional*. When a candidate solution fails, the resulting feedback, such as an environment generated error, or self-generated critique, is informative about the specific reasoning path that produced that failure. If a model is restarted from scratch, it may never revisit the same intermediate states at which the feedback is most relevant. This creates a fundamental mismatch: the supervision is path-specific, but existing inference-time strategies such as sequential refinement largely apply it as though it were a global revision signal.

Tree-structured search methods, such as Tree of Thoughts (Yao et al., 2023), offer an alternative by exploring multiple reasoning paths and backtracking. These methods typically operate at the level of coarse reasoning steps, relying on self-evaluation heuristics to prune branches. Population-based approaches like Recursive Self-Aggregation (Venkatraman et al., 2025) refine solutions by aggregating information across a pool of candidates. Both families of methods improve over flat sampling, but neither leverages fine-grained, token-level signals about where reasoning went wrong.

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

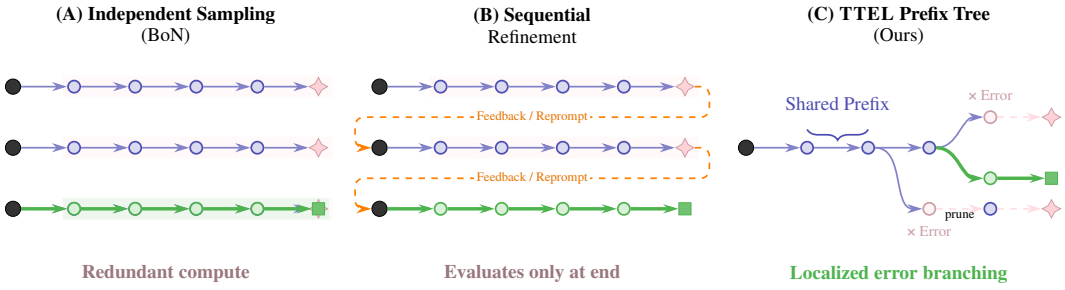


Figure 1: **Search Tree Typology Comparison.** (A) Independent Sampling generates entirely separate full-length trajectories, leading to high computational waste from parallel exploration of similar paths. (B) Sequential Refinement methods link full trajectories end-to-end, evaluating only at terminal steps. (C) TTEL structures search as a prefix-sharing tree, explicitly capitalizing on token-level error localization to branch exactly where reasoning diverges, avoiding redundant computation of shared prefixes and averting terminal failure.

Essentially, these existing approaches either lose the state on which feedback is informative, or preserve the state without extracting a precise corrective signal from it.

This suggests a more targeted view of inference-time improvement: rather than treating feedback as a generic instruction for a fresh retry, can a model use feedback to localize error *on the specific trajectory that produced it*, retain the useful prefix of that trajectory, and selectively regenerate only the suffix where the reasoning becomes suspect? Motivated by this question, we investigate the following core problem:

> *Can an LLM perform token-level credit assignment on its own failed trajectories, and leverage this localized signal to guide test-time scaling optimally?*

We propose TTEL, an inference-time search algorithm that performs token-level error localization without gradient updates. When a generated solution fails, we prompt the model with the environmental feedback to re-evaluate the original trajectory via prompt log-probabilities. By comparing token probabilities before and after observing this feedback, we identify positions of maximal disagreement. A sharp drop in probability indicates that the model strongly rejects its prior decision at that specific step.

To ensure these probability shifts are genuinely driven by the feedback rather than generic context changes, we filter the signal against a baseline with non-diagnostic feedback. TTEL then uses this localized signal to truncate the reasoning trace at the highest-confidence error and branch a new generation from that point. By directing computation toward correcting specific mistakes rather than regenerating entire solutions, this feedback-guided tree search achieves substantially higher token efficiency than standard sampling baselines across diverse reasoning tasks, including competitive programming (Jain et al., 2024) and complex mathematical reasoning (Balunovic et al., 2025).

1.1 Our Contributions.

Our paper shows that LLMs can error correct failing trajectories at inference time based on feedback, and use it to guide test-time search. Specifically:

1. Token-Level Error Localization: We introduce TTEL, an inference-time algorithm that uses probability contrasts by comparing informed environmental feedback against non-diagnostic feedback to localize errors during generation.

2. Efficient Test-Time Search: By coupling local credit assignment with trajectory retention, TTEL truncates and branches only the faulty suffix of a trajectory. This maximizes the reuse of valid reasoning prefixes and eliminates the redundant exploration.

3. Empirical Efficiency: We establish strictly dominating Pareto frontiers across math benchmarks like AIME-2025, and coding benchmarks like LiveCodeBench where TTEL attains pass@64 of 71.0% at roughly half the token cost of independent sampling.

2 Related Work

We provide an abbreviated overview of related work below; an extended discussion mapping TTEL to the broader literature is available in Appendix A.

Test-Time Scaling and Tree Search. Reasoning performance scales reliably with inference-time compute (Snell et al., 2024; DeepSeek-AI, 2025), typically through robust independent sampling arrays (Wang et al., 2023). To iterate beyond flat sampling, sequential refinement methods prepend execution feedback to prompt self-correction (Madaan et al., 2023; Klissarov et al., 2026). However, because they treat trajectory-specific feedback as a global instruction, these models often regenerate reasoning from scratch, triggering context bloat or hallucinated logic (Kumar et al., 2024). Structured backtracking algorithms like Tree of Thoughts (Yao et al., 2023) and population-based search (Venkatraman et al., 2025) remedy this by exploring partial branches, but they rely on programmatic chunking and external heuristics rather than internal, token-level signals.

On-Policy Distillation and Preference Optimization. Recent work has increasingly utilized natural language assessment for dense credit assignment. By evaluating the ratio between a base policy and a feedback-conditioned teacher, methods like Generalized Knowledge Distillation (Agarwal et al., 2024) and On-Policy Self-Distillation (Zhao et al., 2026) optimize student models directly on their own generated rollouts. This core mathematical insight, minimizing the Kullback-Leibler divergence driven by $\pi_\theta(\cdot \mid x)/\pi_\theta(\cdot \mid x, f)$, has established the foundation for new reinforcement learning algorithms, most notably Self-Distillation Policy Optimization (SDPO; Hübotter et al. 2026) and RLTF (Song et al., 2026). Similar mechanisms have also been applied successfully in experiential context distillation (OPCD; Ye et al. 2026) and continual learning (SDFT; Shenfeld et al. 2026).

Repurposing the Signal for Inference-Time Search. Crucially, this existing body of work restricts the application of the $\pi_\theta(\cdot \mid x)/\pi_\theta(\cdot \mid x, f)$ divergence to offline gradient updates, yielding static policy networks. The direct capability of this ratio to localize errors purely as an online, inference-time heuristic has remained fundamentally unaddressed. TTEL bridges this gap by repurposing the token-level signal used in the SDPO framework. Specifically, TTEL extracts an organic, native pruning criterion that structures and bounds test-time search without requiring parameter updates.

3 Preliminaries and Problem Formulation

Let π_θ denote a pre-trained autoregressive language model over vocabulary $\mathcal{V}$. Given a problem prompt x, the model generates a trajectory $y = (y_1, \ldots, y_N)$ with token probabilities

$$p_t^{(S)} := \pi_\theta(y_t \mid x, y_{<t}), \qquad y_{<t} := (y_1, \ldots, y_{t-1}). \tag{1}$$

For a realized trajectory y, we refer to $p_t^{(S)}$ as the *student token probability*, i.e., the probability assigned to the generated token y_t during the original rollout.

We assume access to an execution environment $\mathcal{E}$, which evaluates a candidate trajectory y on problem x and returns a textual feedback string f.

The central challenge in test-time scaling is a *credit assignment bottleneck*: when a generated trajectory fails, the environment evaluates the trajectory globally, but does not identify the specific token position at which the reasoning first became inconsistent with a successful solution. Standard best-of-K sampling discards failed trajectories entirely, while sequential refinement conditions on feedback but does not explicitly localize where to revise the trajectory. Our goal is to use the model itself to perform token-level post-hoc error localization on a failed trajectory, and then use that localized signal to guide branching during inference-time search.

4 Methodology

TTEL operates on a single pre-trained language model π_θ that serves as both the generator and the evaluator of its own outputs. The core mechanism computes the same token-level divergence signal that train-time self-distillation algorithms use to update model weights (Hübotter et al., 2026), but instead of aggregating this signal into a gradient step, TTEL uses it to prune and branch an inference-time search tree. We formalize the algorithm in two core components: detecting and filtering token-level error signals (spikes), and utilizing those signals to guide a search and branching strategy over candidate solutions.

4.1 Spike Detection and Filtering

Given a prompt x, the model first generates a candidate trajectory $y = (y_1, \ldots, y_N)$ under standard autoregressive decoding. For this realized trajectory, we record the *student* token probabilities

$$p_t^{(S)} := \pi_\theta(y_t \mid x, y_{<t}). \tag{2}$$

We obtain textual feedback f and then re-score the *same* token sequence y under a feedback-augmented context, yielding the *teacher* token probabilities

$$p_t^{(T)}(f) := \pi_\theta(y_t \mid x, y_{<t}, f). \tag{3}$$

Note: For notational simplicity, $\pi_\theta(y_t \mid x, y_{<t}, f)$ denotes the probability assigned to token y_t when the original trajectory is re-scored under feedback f. Here, $y_{<t}$ is the original trajectory prefix preceding y_t.

We define the raw feedback-conditioned spike at token t as

$$\Delta_t(f) := p_t^{(S)} - p_t^{(T)}(f). \tag{4}$$

A large positive value of $\Delta_t(f)$ indicates that, after observing feedback f, the model assigns substantially lower probability to its original token choice y_t than it did during generation.

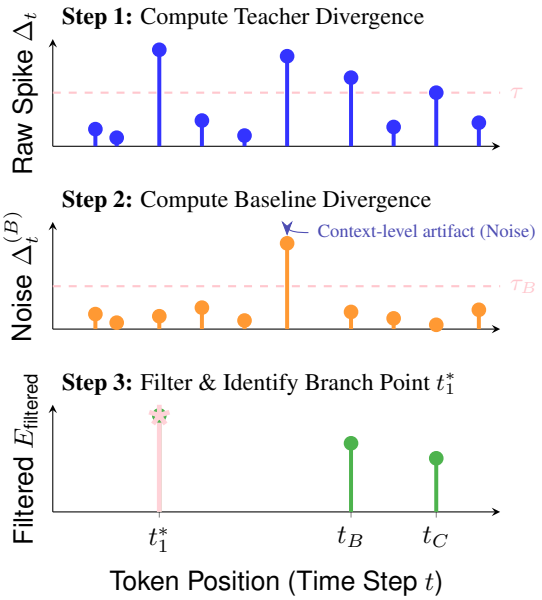


Figure 2: **Token-Level Error Localization.** TTEL filters raw feedback-induced probability drops (blue) against a baseline context with non-diagnostic feedback (orange) to distinguish genuine errors from context artifacts. The maximal filtered spike (green) is selected as the precise branch point t_1^*.

However, this raw spike can reflect two distinct effects: (i) genuine semantic re-evaluation induced by task-relevant feedback, and (ii) generic probability shifts caused by appending additional text to the context window. To control for the latter, we construct non-diagnostic feedback f_B by replacing the re-attempt instruction in f with "Repeat the previous attempt word by word, but skip portions of redundant thinking". We then compute the corresponding *baseline* token probabilities

$$p_t^{(B)}(f_B) := \pi_\theta(y_t \mid x, y_{<t}, f_B). \tag{5}$$

This yields the baseline spike

$$\Delta_t(f_B) := p_t^{(S)} - p_t^{(B)}(f_B). \tag{6}$$

We then define the *filtered* spike score

$$g_t := \Delta_t(f) - \Delta_t(f_B) = p_t^{(B)}(f_B) - p_t^{(T)}(f). \tag{7}$$

This score isolates feedback-specific disagreement by subtracting out generic context-induced shifts that also arise under non-diagnostic feedback.

We first select positions where the teacher's most-likely token differs from the student's original token:

$$\mathcal{D} = \left\{ t \in \{1, \ldots, N\} \;\middle|\; y_t \neq \hat{y}_t^{(T)} \right\}, \qquad \hat{y}_t^{(T)} := \arg\max_v \pi_\theta(v \mid x, y_{<t}, f). \tag{8}$$

Among these, we retain token positions whose teacher spike exceeds a threshold while the baseline spike remains small:

$$E_{\text{filtered}} = \{ t \in \mathcal{D} \mid \Delta_t(f) > \tau \;\wedge\; \Delta_t(f_B) \leq \tau_B \}. \tag{9}$$

4.2 Search and Branching Strategy

Once the localized error set E_{filtered} is computed, the algorithm must choose where to branch. We select the branch point as the token with the strongest feedback-specific disagreement:

$$t^* = \arg\max_{t \in E_{\text{filtered}}} g_t. \tag{10}$$

The search tree then truncates the failed trajectory at position $t^* - 1$ and launches a new generation from the retained prefix $y_{<t^*}$. This mechanism reuses the portion of the trajectory that precedes the strongest localized error signal, rather than discarding the full generation and restarting from scratch.

When $E_{\text{filtered}} = \varnothing$, the available feedback does not localize any actionable token-level error. In this case, the algorithm performs a restart from the original prompt x.

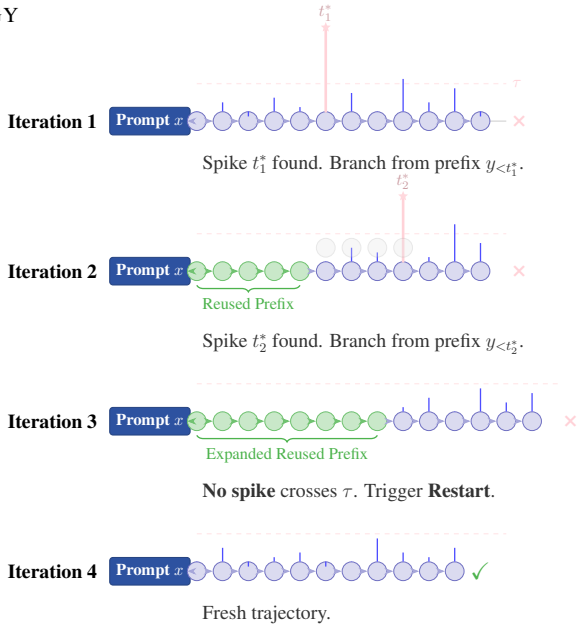


Figure 3: **TTEL Search and Branching Dynamics.** The reasoning trajectory iteratively evolves to isolate errors. At Iteration 1, the model detects a filtered error spike t_1^* (red) and branches a new generation from the retained prefix $y_{<t_1^*}$ (green). If a trajectory fails without a discernible spike (e.g., Iteration 3 where $E_{\text{filtered}} = \varnothing$), the tree search triggers a restart from the root prompt.

5 The TTEL Algorithm

Algorithm 1 summarizes the full TTEL procedure. The search tree $\mathcal{T}$ stores *retained prefixes* of previously generated trajectories, with the root corresponding to the empty prefix. At each iteration, the algorithm selects a leaf prefix $y_{<t}$ from $\mathcal{T}$ and generates a full candidate continuation conditioned on both the original problem prompt x and the retained prefix. The resulting trajectory is then evaluated by the execution environment $\mathcal{E}$, which returns feedback f for re-scoring and subsequent regeneration. The diagnostic component of f may contain environment-derived information, such as a compiler error, runtime error, or failing public test case.

The algorithm performs token-level spike detection by re-scoring the same trajectory under two alternative contexts: a feedback-conditioned context using the actual feedback f, and a baseline context using the non-diagnostic feedback f_B. This yields a feedback-conditioned spike score $\Delta_t(f) = p_t^{(S)} - p_t^{(T)}(f)$ and a baseline spike score $\Delta_t(f_B) = p_t^{(S)} - p_t^{(B)}(f_B)$ at each token position. The filtered spike score

$$g_t = \Delta_t(f) - \Delta_t(f_B)$$

captures disagreement that is specific to the observed failure signal rather than generic context augmentation.

The algorithm then constructs the localized error set

$$E = \{\, t \mid \Delta_t(f) > \tau \;\wedge\; \Delta_t(f_B) \le \tau_B \;\wedge\; y_t \neq \hat{y}_t^{(T)} \,\},$$

which contains token positions where the student's original token disagrees with the teacher's most-likely token, and whose probability drops sharply under true feedback but not under non-diagnostic feedback. If this set is nonempty, TTEL selects the branch point $t^* = \arg\max_{t \in E} g_t$ and adds the retained prefix $y_{<t^*}$ to the tree, provided $t^* \le D_c$. A subsequent generation from this prefix reuses the earlier portion of the trajectory while revising the region where the model exhibits the strongest feedback-specific disagreement. If no actionable spike is detected, or if the best spike lies beyond the depth cap D_c, the algorithm performs a restart by adding the empty prefix back into the tree, thereby encouraging continued exploration from scratch rather than overcommitting to an uninformative failed trajectory.

The procedure repeats until the inference-time budget K is exhausted, where the budget may be measured in generated tokens, completed trajectories, or environment calls. The procedure returns the generated candidate set $\mathcal{S}$, which is evaluated using pass@k. An external selector, such as a verifier score, execution outcome, or majority-based aggregation, may optionally be applied when a single answer is required. In this way, TTEL converts token-level self-re-evaluation into an explicit tree-search policy that adaptively allocates computation toward revising localized errors instead of repeatedly regenerating entire solutions.

Alg. 1 TTEL: Feedback-Guided Tree Search

Require: Prompt x, model π_θ, env. $\mathcal{E}$, thresholds τ, τ_B, budget B, depth cap D_c

$\mathcal{T} \leftarrow \{\varnothing\}$ `// retained prefixes`
$\mathcal{S} \leftarrow \varnothing$ `// generated candidates`
$f \leftarrow \varnothing$ `// no feedback initially`
while tokens generated $< B$ **do**
  Select leaf $y_{<t} \in \mathcal{T}$
  $y \leftarrow \textsc{Generate}(\pi_\theta, x, y_{<t}, f)$
  $f \leftarrow \mathcal{E}(x, y)$ `// feedback`
  ▷ *Spike Detection*
  $\Delta_t(f) \leftarrow p_t^{(S)} - p_t^{(T)}(f)$
  $\Delta_t(f_B) \leftarrow p_t^{(S)} - p_t^{(B)}(f_B)$
  $\mathcal{D} \leftarrow \{t \mid y_t \neq \hat{y}_t^{(T)}\}$
  $E \leftarrow \{t \in \mathcal{D} \mid \Delta_t(f) > \tau \wedge \Delta_t(f_B) \leq \tau_B\}$
  $t^* \leftarrow \arg\max_{t \in E}(\Delta_t(f) - \Delta_t(f_B))$
  ▷ *Branching*
  if $E \neq \varnothing \ \wedge \ t^* \leq D_c$ **then**
    Add $y_{<t^*}$ to $\mathcal{T}$ `// branch`
  else
    $\mathcal{T} \leftarrow \{\varnothing\}$
    $f \leftarrow \varnothing$
  end if
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{y\}$
end while
return $\mathcal{S}$

6 Experiments

In this section, we outline our experimental framework designed to evaluate the effectiveness of our proposed method, TTEL. We focus our evaluation on two challenging domains demanding multi-step planning: complex mathematical reasoning and code generation. Our empirical evaluation is designed to answer the following core research questions (RQ):

- **RQ1 (Compute Scalability & Generalization):** How efficiently does TTEL scale inference-time compute to uncover correct trajectories compared to standard multi-turn paradigms, and does this scaling advantage generalize across environments with and without explicit verification feedback?
- **RQ2 (The Role of Context in Error Localization):** How strictly does accurate token-level credit assignment depend on the availability of rich environment execution signals and the retention of the historical reasoning trace?
- **RQ3 (Filtering Noise vs. Semantic Signal):** Is the baseline subtraction strictly necessary to isolate genuine, feedback-driven model corrections from generic context-induced probability shifts?

Evaluation Benchmarks To rigorously assess the capabilities of our approach, we focus on the following two benchmark categories:

- **Mathematical Reasoning:** We evaluate on **AIME-25** dataset, and **HMMT-25** from the MathArena benchmark (Balunovic et al., 2025). Each dataset consists of 30 highly challenging, competition-level mathematics problems designed to test deep, multi-step analytical reasoning without intermediate execution feedback.
- **Code Generation:** We utilize **LiveCodeBench** (Jain et al., 2024), specifically a recent subset of 131 contest-style coding questions released between February and May 2025. This benchmark optionally provides rigorous execution feedback via public test cases.

We define two variants for our method based on the feedback:

- **TTEL-GenFB:** This uses generic feedback which includes previous answers, the full reasoning trace for the last attempt, a static error message ("Your previous attempt was unsuccessful.") and a re-attempt instruction. This is applicable for all benchmarks.
- **TTEL-EnvFB:** The static error message in TTEL-GenFB is replaced by rich environment feedback such as runtime errors or failing public test cases. This can be tested on LiveCodeBench as it provides execution based feedback, but not on maths benchmarks.

Implementation Details We utilize the **Qwen3-8B** and **Qwen3-4B-Thinking-2507** models for our experiments. To compare inference-time scaling across methods, we report both the *rollout budget* k and the *total number of generated tokens*. The maximum number of newly generated tokens per rollout is capped at 16,384. Decoding across all methods uses a temperature $T = 1$ and $\text{top}_p = 1$. For

TTEL, we use a spike detection threshold of $\tau = 0.06$ and limit the retained prefix to a maximum checkpoint depth of $D_c = 16{,}384$ tokens. To standardize evaluation, our primary metric is **pass@**k, measuring whether at least one of the k adaptively generated rollouts produces a correct solution.

Baselines To isolate the performance gains of TTEL, we compare our method against three robust test-time scaling baselines:

1. **Independent Sampling:** A standard parallel generation (best-of-k) approach. The model independently generates k full rollouts, and we evaluate whether any trajectory successfully solves the problem.
2. **Multi-turn Refinement:** A sequential feedback-driven approach. The model is allowed a maximum of k interactive turns to arrive at the correct answer, where each subsequent attempt is conditioned on the full sequence of previous answers and their respective feedback.
3. **Recursive Self-Aggregation (RSA):** A recent population-based test-time scaling algorithm that searches through a parameter space characterized by population size N, aggregation size K, and a distinct number of sequential steps T.

7 Experimental Results

We structure our results to directly address the research questions outlined in Section 6. We first analyze the generation-cost optimal scaling capabilities of TTEL across domains (RQ1), followed by detailed ablations dissecting how structural context (RQ2) and baseline filtering using non-diagnostic feedback (RQ3) drive its error localization precision.

7.1 Compute-Optimal Scaling and Generalization (RQ1)

We evaluate TTEL and our selected baselines across a varying rollout budget k to analyze the scaling dynamics of pass@k relative to overall token efficiency. We do not assume access to environment feedback and utilize the same generic error message as part of feedback in TTEL-GenFB and Multi-Turn Refinement.

Scaling on LiveCodeBench. The top row of Figure 4 illustrates the Pareto frontier of pass@k relative to the average token cost per question on the LiveCodeBench V6 subset. We omit environment feedback (runtime errors, failing test cases) to demonstrate the effectiveness of TTEL with just generic feedback. TTEL establishes a strictly dominating Pareto frontier, achieving a superior pass@k rate for any given generated-token budget. At a fixed maximum rollout budget of $k = 64$, TTEL achieves a pass@64 of 0.71. Notably, it outperforms both multi-turn refinement (which stagnates at 0.567) and independent sampling (0.646), while consuming less than half the average tokens of independent sampling. This compute efficiency validates that branching from retained prefixes prevents redundant regeneration. Note that the shaded region denotes the standard deviation for that point over 3 seeds.

Generalization to Math. We further evaluate scaling on the AIME-25 and HMMT-25 benchmarks, where the environment only provides terminal failure states without intermediate diagnostic feedback. The middle and bottom rows of Figure 4 illustrate the corresponding Pareto frontiers on AIME-25 and HMMT-25. At a fixed maximum rollout budget of $k = 64$, TTEL achieves a pass@64 of 0.833 on AIME-25 and 0.608 on HMMT-25 with Qwen3-8B. With Qwen3-4B-Thinking-2507, it achieves a pass@64 of 0.850 on AIME-25 and 0.692 on HMMT-25. In each setting, TTEL attains the highest pass@64 among all methods. This demonstrates that using generic feedback induces actionable token-level probability shifts, confirming that TTEL operates as a domain-agnostic test-time scaling method. Note that the shaded region denotes the standard deviation for that point over 4 seeds.

7.2 The Role of Context in Error Localization (RQ2)

Accurate token-level credit assignment depends on the context provided to the teacher during spike detection. We hypothesize that retention of the original thought trace y, and the specificity of the external feedback f can enable more precise error-localization.

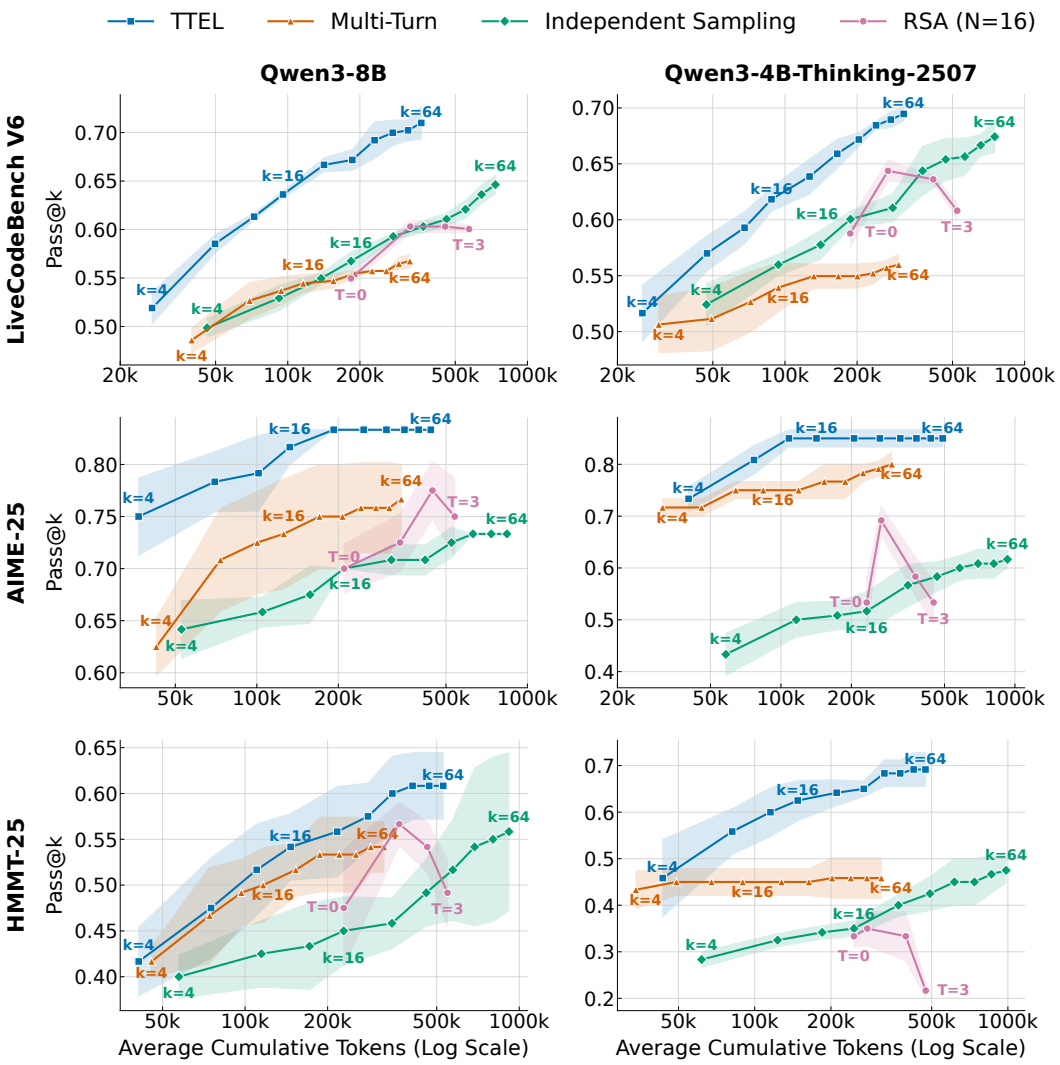


Figure 4: Pareto plot of pass@k vs. average tokens generated per question on LiveCodeBench V6 (top), AIME-25 (middle) and HMMT-25 (bottom). Shaded regions denote the standard deviation across seeds (3 for LCBv6, 4 for AIME-25 and HMMT-25).

As illustrated in Figure 5, blinding the teacher to the *Full Trace* generated by the student and conditioning execution failure solely on the extracted final answer causes a severe drop in pass@k (from 0.630 to 0.597 in case of *TTEL-GenFB*) alongside inflated token consumption. Accurate mistake localization algebraically requires the model to review the logic that actually produced the error, rendering the *Full Trace* strictly necessary. Furthermore, advancing beyond generic feedback by injecting explicit execution traces aggressively grounds the probability distribution. This explicit external signal triggers far more precise log-probability divergences than implicit self-correction, lowering search cost and boosting pass@k to 0.658 for *TTEL-EnvFB*. Ultimately, token-efficient search relies on maximizing diagnostic context f during re-scoring.

7.3 Isolating Semantic Signals via Baseline Filtering (RQ3)

Finally, we validate the core architectural component of TTEL: the baseline filter, $\Delta_t(f_B)$. While differences in student and teacher probabilities can identify divergence in reasoning due to genuine mistakes, the updated context for the teacher also induces a baseline probability shift. For example, the teacher may try to rephrase or abridge the reasoning trace of the student, leading to a large number

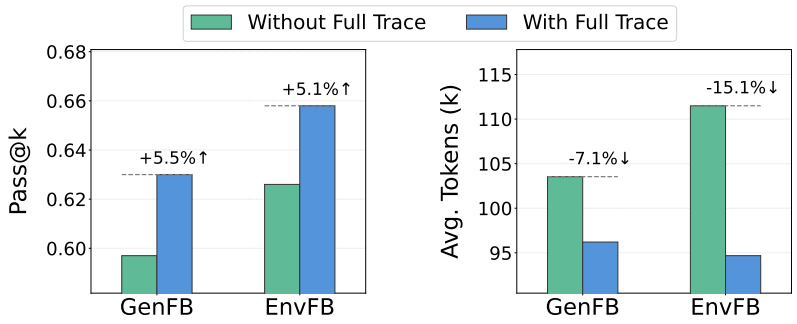


Figure 5: The impact of context ablation on LiveCodeBench V6 for Qwen3-8B ($k = 16$), averaged over 4 seeds. We evaluate both TTEL-GenFB and TTEL-EnvFB to measure the impact of informative environment feedback. To demonstrate the necessity of Full Thought Trace from the previous attempt, we include ablations that utilize only final answers from previous attempts.

of false-positive spikes that are unsuitable for diverging into novel branches. This is illustrated in section G in the Appendix.

Impact of Baseline Spike Filtering. This ablation utilizes raw teacher probability spikes $\Delta_t(f)$ to trigger branching, entirely removing the secondary baseline subtraction. As shown in Figure 6, removing this filter dramatically changes the search dynamics. The average number of detected spikes in Turn 1 explodes from 19.3 to 486.0, representing massive false-positive mistake detections driven merely by context augmentation rather than genuine error correction. This noisy search degrades downstream pass@k to 0.592. Thus, the baseline filter is strictly required to normalize the signal, filter context-induced probability shifts, and accurately localize actionable mathematical or logical errors.

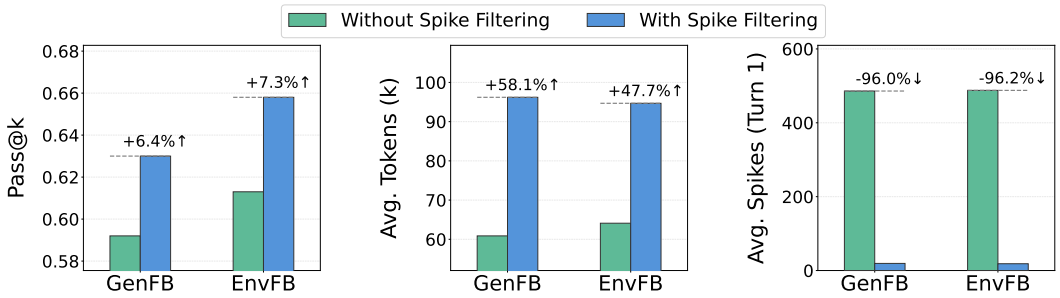


Figure 6: Baseline removal leads to massive false-positive spike detections and lowers accuracy on LiveCodeBench V6 for Qwen3-8B ($k = 16$), averaged over 4 seeds.

8 Conclusion:

In this work, we introduced Test-Time Scaling via Error Localization (TTEL), demonstrating that LLMs can perform precise token-level credit assignment at inference time without gradient updates. By coupling localized error detection directly with trajectory retention, TTEL maximally reuses valid reasoning prefixes and mitigates computational redundancy. Our results establish new generation-cost optimal Pareto frontiers, across math and coding tasks.

Ethics Statement

This work proposes an inference-time algorithm for improving reasoning in LLMs. We do not train or fine-tune models, introduce new datasets, or collect human data. All benchmarks used (LiveCodeBench, AIME-25, HMMT-25) are publicly available. We do not foresee specific ethical concerns beyond those inherent to improving LLM capabilities.

Reproducibility Statement

All hyperparameters, decoding settings, and prompt templates are specified in Section 6 and Appendix B. We use publicly available models (Qwen3-8B and Qwen3-4B-Thinking-2507) and benchmarks. Detailed per-seed results for all experiments are provided in Appendix F. We report mean and standard deviation across seeds (3 for LiveCodeBench, 4 for AIME-25 and HMMT-25) for all metrics.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The twelfth international conference on learning representations*, 2024.

Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating LLMs on uncontaminated math competitions. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=y0zL9IZxZ7.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*, 2024.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Shizhe Diao, Pengcheng Wang, Yong Lin, Rui Pan, Xiang Liu, and Tong Zhang. Active prompting with chain-of-thought for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1330–1350, 2024.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*, 2022.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pp. 10764–10799. PMLR, 2023.

Taneesh Gupta, Rahul Madhavan, Rishabh Tiwari, Xuchao Zhang, Chetan Bansal, Saravan Rajmohan, and Kurt Keutzer. Adaptive control for test-time scaling. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*, 2025a.

Taneesh Gupta, Shivam Shandilya, Xuchao Zhang, Rahul Madhavan, Supriyo Ghosh, Chetan Bansal, Huaxiu Yao, and Saravan Rajmohan. Carmo: Dynamic criteria generation for context aware reward modelling. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 2202–2261, 2025b.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457*, 2024.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Martin Klissarov, Jonathan Cook, Diego Antognini, Hao Sun, Jingling Li, Natasha Jaques, Claudiu Musat, and Edward Grefenstette. Improving interactive in-context learning from natural language feedback. *arXiv preprint arXiv:2602.16066*, 2026.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Tianqi Liu, Wei Xiong, Jie Ren, Lichang Chen, Junru Wu, Rishabh Joshi, Yang Gao, Jiaming Shen, Zhen Qin, Tianhe Yu, et al. Rrm: Robust reward model training mitigates reward hacking. *arXiv preprint arXiv:2409.13156*, 2024.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Yuda Song, Lili Chen, Fahim Tajwar, Remi Munos, Deepak Pathak, J Andrew Bagnell, Aarti Singh, and Andrea Zanette. Expanding the capabilities of reinforcement learning via text feedback. *arXiv preprint arXiv:2602.02482*, 2026.

Pragya Srivastava, Harman Singh, Rahul Madhavan, Gandharv Patil, Sravanti Addepalli, Arun Suggala, Rengarajan Aravamudhan, Soumya Sharma, Anirban Laha, Aravindan Raghuveer, et al. Robust reward modeling via causal rubrics. *arXiv preprint arXiv:2506.16507*, 2025.

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. *arXiv preprint arXiv:2211.14275*, 2022.

Siddarth Venkatraman, Vineet Jain, Sarthak Mittal, Vedant Shah, Johan Obando-Ceron, Yoshua Bengio, Brian R Bartoldson, Bhavya Kailkhura, Guillaume Lajoie, Glen Berseth, et al. Recursive self-aggregation unlocks deep thinking in large language models. *arXiv preprint arXiv:2509.26626*, 2025.

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)*, pp. 9426–9439, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, and Quoc V Le. H. chi, sharan narang, aakanksha chowdhery, and denny zhou. self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, volume 1, pp. 2, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026.

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

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

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. *arXiv preprint arXiv:2310.04406*, 2023.

SUPPLEMENTARY MATERIALS

A EXTENDED RELATED WORK

Inference-time scaling via sampling, reranking, and verifier selection. The capability of large language models on complex reasoning and programming tasks scales predictably with inference-time compute (Snell et al., 2024; Brown et al., 2024; Wu et al., 2024; Muennighoff et al., 2025; DeepSeek-AI, 2025). The most straightforward and robust baseline in this family is independent parallel sampling, where massive arrays of complete candidates are generated and evaluated under specific scaling laws (Brown et al., 2024). Techniques such as pass@k, Best-of-K, majority voting, and verifier reranking (Lightman et al., 2023) attempt to select the optimal solution from this generated pool. While these methods reliably convert increased test-time compute into higher accuracy, they are fundamentally limited by redundancy: they generate multiple full candidates entirely independently, failing to reuse trajectory-specific feedback or preserve valid partial progress from failed attempts.

Chain-of-thought prompting and structured reasoning elicitation. Chain-of-Thought (CoT) prompting (Wei et al., 2022) revealed that eliciting explicit intermediate reasoning steps dramatically improves language model performance on complex multi-step tasks. Self-Consistency (Wang et al., 2023) extends this paradigm by sampling diverse CoT reasoning paths and aggregating via majority vote, effectively combining qualitative reasoning structure with parallel sampling. Further refinements include complexity-based prompting, which preferentially selects more elaborate reasoning chains (Fu et al., 2022); Program-of-Thought prompting, which externalizes computational steps to code interpreters (Chen et al., 2022; Gao et al., 2023); and active prompting, which strategically selects the most informative exemplars to elicit stronger chains (Diao et al., 2024). These methods collectively establish that the *structure* and *quality* of reasoning is often the critical determinant of test-time performance. TTEL deepens this insight: rather than eliciting better reasoning through prompt engineering, it uses a feedback-conditioned token-level signal to diagnose precisely *where* reasoning quality degrades and surgically branches from that exact locus.

Sequential refinement and interactive feedback at inference time. To address the inefficiency of independent sampling, sequential refinement methods prepend execution feedback to prompt the model to iteratively correct its own errors (Madaan et al., 2023). Recent work investigates interactive in-context learning from natural language feedback as a distinctly trainable capability (Klissarov et al., 2026). While these approaches successfully leverage environmental critiques, the setting remains fundamentally multi-turn in-context learning. Treating trajectory-specific feedback as a global instruction for a sequential retry forces the model to regenerate the reasoning chain from scratch, frequently resulting in context bloat, hallucination, or the repetition of earlier logical errors (Kumar et al., 2024). TTEL mitigates this by using feedback not for an unanchored sequential retry, but for prefix-anchored branching.

Tree search and structured exploration over reasoning traces. To overcome the limitations of sequential generation, backtracking algorithms such as Tree of Thoughts (Yao et al., 2023) and Language Agent Tree Search (Zhou et al., 2023) explicitly structure inference as a search space over partial reasoning trajectories. Population-based and evolutionary methods, including Recursive Self-Aggregation (Venkatraman et al., 2025) and AlphaEvolve (Novikov et al., 2025), scale this by iteratively mutating and recombining successful reasoning paths to discover complex solutions. These methods establish language model inference as a formal search procedure. Concurrently, inference-time control algorithms explicitly optimize computational budgets utilizing explicit constraints, either by deploying verifiable testing mechanisms and adaptive early-stopping criteria to dynamically bound test-time horizons (Gupta et al., 2025a), or by applying simpler sequential budget-forcing sequences (Muennighoff et al., 2025). However, a primary computational bottleneck across these tree-search and explicit control frameworks remains their reliance on manual, coarse-grained chunking, heuristic bounds, or whole-trajectory mutation: they divide generation into heuristically defined "steps" and

evaluate states using external value functions. TTEL is fundamentally a tree-search method, but one whose branching rule emerges organically from feedback-conditioned token disagreement rather than hand-designed programmatic search heuristics or blind mutation.

Verifier-guided reasoning, execution feedback, and program repair. In domains such as competitive programming and mathematical reasoning, execution environments provide rich diagnostic feedback, including runtime errors and public test case outputs (Jain et al., 2024). Previous literature in automated program repair and verifier-guided generation uses this execution feedback primarily to score, rerank, or prompt complete revisions of generated candidates. A key contrast of our work is that TTEL utilizes verifier feedback not merely to evaluate candidates, but to explicitly decide *where* to truncate the trajectory and branch a new attempt.

Process reward models and step-level credit assignment. Outcome reward models assign a single scalar to complete solutions, limiting their capacity for localizing errors within multi-step reasoning traces. Process reward models (PRMs) address this gap by providing step-level supervision, either through human annotation (Lightman et al., 2023; Uesato et al., 2022) or automated Monte Carlo estimation procedures such as Math-Shepherd (Wang et al., 2024) and OmegaPRM (Luo et al., 2024). Causal reward modelling techniques such as CROME (Srivastava et al., 2025), RRM (Liu et al., 2024) and CARMO (Gupta et al., 2025b) also attempt to address credit assignment through causal perturbations. The key limitation is that PRMs require training, introducing additional annotation costs and potential distribution mismatch when applied to new domains or model families. TTEL achieves analogous credit assignment, but derives its signal purely from the probabilistic disagreement between the base and feedback-conditioned policies, requiring no externally trained reward model.

On-policy distillation and learning from self-generated trajectories. Because environment verifiers typically provide only terminal outcomes, minimizing the distribution mismatch between training and inference requires on-policy algorithms. Generalized Knowledge Distillation (GKD) trains the student model on its own generated output sequences while leveraging feedback from the teacher, preventing off-policy divergence (Agarwal et al., 2024). Similarly, On-Policy Self-Distillation (OPSD) utilizes a single model acting as both teacher (conditioned on privileged verified reasoning traces) and student, minimizing the divergence over the student's own rollouts (Zhao et al., 2026). These highly relevant methods learn directly from the model's own trajectories, but they internalize the signal into model parameters through offline training, rather than dynamically guiding inference-time branching on the current problem instance.

Self-distillation and RL from rich textual feedback. Recent advancements have formalized reinforcement learning protocols capable of exploiting rich, textual environment feedback to overcome the scalar credit assignment bottleneck. Self-Distillation Policy Optimization (SDPO) converts tokenized feedback into a dense learning signal by treating the feedback-conditioned model as a self-teacher, distilling its next-token predictions back into the policy (Hübotter et al., 2026). Concurrently, RLTF explores self-distillation and feedback-modeling objectives to internalize multi-turn textual critique for improved single-turn test-time performance (Song et al., 2026). Conceptually, these works are the closest to our approach, as they utilize textual critique for dense token-level credit assignment. However, they primarily improve the policy through offline training. TTEL demonstrates that similar token-level divergence signals can be repurposed to guide search organically *without* gradient updates.

Training for extended reasoning via reinforcement learning. A parallel line of work focuses on training models to produce longer, more deliberate reasoning traces. STaR (Zelikman et al., 2022) and V-STaR (Hosseini et al., 2024) bootstrap reasoning ability by iteratively fine-tuning on self-generated correct rationales. At larger scale, Gemini 2.5 (Comanici et al., 2025) employs reinforcement learning to produce extended internal chains of thought, while DeepSeek-R1 (DeepSeek-AI, 2025) demonstrates that RL with verifier rewards can elicit sophisticated emergent behaviors including self-verification and backtracking. GRPO (Shao et al., 2024) and related group-relative policy optimization methods reduce the need for explicit critic models in this setting. These training-time investments produce reasoning models whose outputs are naturally amenable to test-time scaling interventions. TTEL is complementary to this paradigm: it operates purely at inference time on any model capable of generating reasoning traces, requiring no additional training or parameter updates,

and can be composed with RL-trained reasoning models to further improve their test-time search efficiency.

Context distillation and internalizing auxiliary information into weights. Context distillation methods aim to convert useful but cumbersome context, such as optimized system prompts, retrieval augmentations, or historical traces, directly into model parameters. For instance, On-Policy Context Distillation (OPCD) bridges on-policy learning with context distillation by training student models on generated trajectories to match a context-conditioned teacher (Ye et al., 2026). TTEL is related to this paradigm in spirit, as it also leverages the auxiliary information provided by prior attempts and environment feedback. However, rather than absorbing that information offline to permanently update weights, TTEL uses the auxiliary feedback online to evaluate and anchor the search at a securely retained prefix.

Self-distillation beyond post-training: demonstrations and continual learning. The utility of on-policy self-distillation extends beyond reasoning verification. Self-Distillation Fine-Tuning (SDFT) establishes it as a powerful recipe for continual learning from demonstrations, leveraging a demonstration-conditioned model as an on-policy teacher to acquire new skills while avoiding catastrophic forgetting (Shenfeld et al., 2026). While less directly focused on verifier-guided reasoning, SDFT highlights that context-conditioned self-teaching signals are part of a broader, highly effective algorithmic trend. TTEL positions itself within this trend, uniquely distinct by deploying the self-teaching signal purely at inference time.

Overthinking, reasoning efficiency, and adaptive compute allocation. As reasoning models generate increasingly long chains of thought, the problem of *overthinking* expending substantial compute on problems solvable with far less deliberation has emerged as a practical concern (Chen et al., 2024; Gupta et al., 2025a). Adaptive compute allocation methods seek to match reasoning effort to problem difficulty, for instance through early-exit mechanisms, confidence-based stopping criteria, or learned allocation policies. TTEL's spike-based branching mechanism naturally addresses this inefficiency: by localizing errors to specific token positions via feedback-conditioned disagreement, it preserves demonstrably correct reasoning prefixes and avoids redundant regeneration of already-valid computation. This prefix-anchored branching strategy ensures that test-time compute is concentrated precisely where reasoning fails, rather than distributed uniformly or wastefully across the full trajectory.

Summary of Positioning. In summary, TTEL lies at the intersection of verifier-guided inference-time scaling, token-level self-distillation-style credit assignment, and structured tree search. Its distinguishing conceptual feature is that it couples feedback-conditioned credit assignment with retained-prefix branching on the exact failed trajectory. By doing so, it transcends the limitations of methods that utilize feedback exclusively for outcome reranking, unanchored multi-turn retries, or static offline parameter updates.

B Experimental Configuration and Prompt Templates

Generation Settings We experiment with the **Qwen3-8B** and **Qwen3-4B-Thinking-2507** models. The maximum number of newly generated tokens per rollout is capped at 16,384. Decoding across all methods uses a temperature $T = 1$ and $\text{top}_p = 1$.

Test-Time Scaling via Error Localization For TTEL, we use a localized spike detection threshold of $\tau = 0.06$ and $\tau_B = 0.06$ for the baseline. We limit the retained prefix to a maximum checkpoint depth of $D_c = 16{,}384$ tokens. Token probabilities are computed from the top-100 log-probabilities returned per position. Tokens outside this set are assigned a probability of approximately zero.

At the first turn or after a restart, the user prompt is the problem statement. For further turns, the user prompt also includes a history of prior answers, respective feedback and the reasoning trace for the final attempt.

Student Prompt (With History)

{question}
Attempt:
The following is a previous attempt to solve the question:
{answer$_1$} *(answer-only for earlier turns)*
Feedback:
The following is feedback from your unsuccessful earlier attempt:
{feedback$_1$}
Solve the original question. Use the feedback to make changes.

⋮ *(repeat for turns $2 \dots t-1$)*

Attempt:
The following is a previous attempt to solve the question:
{full_trace$_{t-1}$} *(full reasoning+answer for last turn only)*
Feedback:
The following is feedback from your unsuccessful earlier attempt:
{feedback$_{t-1}$}
Solve the original question. Use the feedback to make changes.

Teacher Prompt uses the same format but it includes information from the latest rollout generated by the student. Teacher Prompt at Turn t becomes Student Prompt at Turn t+1.

Spike-Baseline uses the Teacher Prompt but replaces the instruction "Solve the original question. Use the feedback to make changes." in every feedback block with "Repeat the previous attempt word by word, but skip portions of redundant thinking."

Independent Sampling k independent rollouts are sampled from a single prompt. The prompt is simply the problem statement.

Independent Sampling Prompt

{question}

Multi-Turn Refinement This method carries out sequential refinement of answers by conditioning on a history of prior answers and feedback.

Multi-Turn prompt (turn $t > 0$)

{question}
Attempt:
The following is a previous attempt to solve the question:
{non_think_response$_i$} *(answer-only for all turns)*
Feedback:
The following is feedback from your unsuccessful earlier attempt:
{feedback$_i$}
Solve the original question. Use the feedback to make changes.

⋮ *(repeat for all previous turns)*

Recursive Self-Aggregation (RSA) We run RSA with a population of $N = 16$ over $T = 4$ turns and aggregation size $K = 4$. The pass@k values shown in plots are calculated over the $k = 16$ rollouts present in the population after a turn T. At Turn 0, N independent rollouts are generated from the question prompt. At Turns 1 to 3, K candidate answers are sampled from the current population and presented to the model:

RSA aggregation prompt ($K > 1$)

```
You are given a {problem_kind} and several candidate solutions.
Some candidates may be incorrect or contain errors.  Aggregate the
useful ideas and produce a single, high-quality solution.  Reason
carefully; if candidates disagree, choose the correct path.  If
all are incorrect, then attempt a different strategy.  End with the
final result in {format_hint}.

Problem:
{question}

Candidate solutions (may contain mistakes):

---- Solution 1 ----
{answer_1}

⋮

---- Solution K ----
{answer_K}

Now write a single improved solution.  Provide clear reasoning and
end with the final answer in {format_hint}.
```

The `format_hint` adapts by domain. For example, `\boxed{}` for math.

C AI USE DISCLOSURE

In this work, we used generative AI tools for assistance with implementation of our method, baselines, and related scripts, and for improving the readability of the manuscript. We have not used generative AI tools to develop theoretical models or conceptual frameworks, propose or refine hypotheses, design or provide feedback on research methodology or experiments, formulate mathematical claims or proofs, generate synthetic datasets, clean or reformat datasets, interpret results, create or modify scientific figures, or draft sections of the paper, and assisting with translation and supporting qualitative or thematic data analysis are not applicable to this work. We have reviewed all AI-assisted work. For instance, we verified plotting scripts by independently deriving data points from the rendered plots and confirming they matched the underlying data. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

D LIMITATIONS.

TTEL currently explores a single branch per turn. Extending the framework to maintain and expand multiple parallel branches is a future direction. We currently evaluate on domains with objectively verifiable answers (math and code). Exploring such methods on non-verifiable domains is an open direction for future work.

E THEORETICAL ANALYSIS

Notation. Let x be the problem, y a failed trajectory, and f the corresponding feedback, which includes y. Let $c = (x, f)$ denote this shared context. Suppose TTEL selects a branch point t^* and retains the anchor prefix $a = y_{<t^*}$.

We compare two post-failure strategies:

1. Restart: Generating a fresh attempt from the context without anchoring, $Y^{\mathrm{R}} \sim \pi_\theta(\cdot \mid c)$.

2. Branch: Forcing generation to begin from the anchor prefix, $Y^{\mathrm{B}} \sim \pi_\theta(\cdot \mid c, a)$.

Let $V(\hat{y}) \in \{0, 1\}$ denote whether candidate $\hat{y}$ correctly solves the problem.

Theorem 1 (Branching Advantage)**.** Assume *autoregressive prefix consistency*, meaning that freely generating a sequence that begins with a is distributionally equivalent to explicitly anchoring genera-

tion at a:

$$\mathcal{L}\big(Y^{\mathrm{R}}_{\geq|a|+1} \mid Y^{\mathrm{R}}_{<|a|+1} = a,\, c\big) = \mathcal{L}\big(Y^{\mathrm{B}}_{\geq|a|+1} \mid c, a\big).$$

Let $p_a = \Pr(Y^{\mathrm{R}}$ begins with $a \mid c)$ be the probability that a restart naturally recovers the anchor prefix. Let s_a and $s_{\neg a}$ be the probability of success given that the generation does, or does not, begin with a. Then:

$$\Pr\big(V(Y^{\mathrm{B}})=1 \mid c, a\big) - \Pr\big(V(Y^{\mathrm{R}})=1 \mid c\big) \;=\; (1-p_a)(s_a - s_{\neg a}).$$

Proof. By prefix consistency, the success rate of branching is exactly the success rate of restart conditioned on recovering the prefix: $\Pr(V(Y^{\mathrm{B}})=1) = s_a$. By the law of total probability, the unconditional success rate of restart is:

$$\Pr(V(Y^{\mathrm{R}})=1) = p_a s_a + (1-p_a)s_{\neg a}.$$

Subtracting the second equation from the first yields the result. □

The theorem formally justifies the TTEL branching mechanism over standard sequential refinement. Verifier feedback f is trajectory-conditional: it provides correction specifically relevant to the reasoning state at the anchor prefix a. A restart strategy generated from scratch (Y^{R}) may never naturally return to this necessary intermediate state ($p_a \ll 1$), rendering the feedback useless or misleading.

Takeaway

Branching structurally guarantees that the generation is anchored exactly where the feedback's correction becomes actionable. It strictly improves over sequential restart as long as success is more likely once the correct prefix is reached ($s_a > s_{\neg a}$).

F Detailed Results for Scaling Experiments

This section provides detailed results and error analysis for scaling experiments with a varying rollout budget k. Experiments are conducted over 2 models, 3 datasets and repeated over different seeds, denoted by s.

AIME-25 and HMMT-25 have 30 questions each as opposed to LCBv6 with 131 questions. We evaluate the latter with 3 seeds and the former with 4 seeds to account for higher variance.

The reported token count is the average number of tokens generated per question. For RSA, pass@k evaluation is over for a constant $k = 16$, which is the population of $N = 16$ rollouts generated after turn i (denoted by t_i). All other methods scale up to a maximum budget of $k = 64$.

F.1 Total-Token Scaling

Figure 4 reports pass@k against generated (output) tokens only. Since output tokens are decoded autoregressively, they dominate wall-clock latency. Here we additionally plot pass@k against *total* tokens (input + output) in Figure 7, where input tokens correspond to the prefill stage and are processed in parallel.

Token accounting. For Independent Sampling and Multi-Turn, total tokens at budget k sum the prompt and response tokens of every rollout; because neither method shares a KV cache across rollouts, prompt tokens are counted k times. RSA follows the same accounting across its N=16 rollouts at each round T. For TTEL, each turn n contributes the teacher, baseline, and student prefill passes plus the autoregressively decoded student continuation. The teacher prompt at turn $n-1$ is a prefix of the student prompt at turn n, so the overlapping tokens are not double-counted.

Across all six model–benchmark settings, TTEL retains the highest pass@64 under this total-token metric.

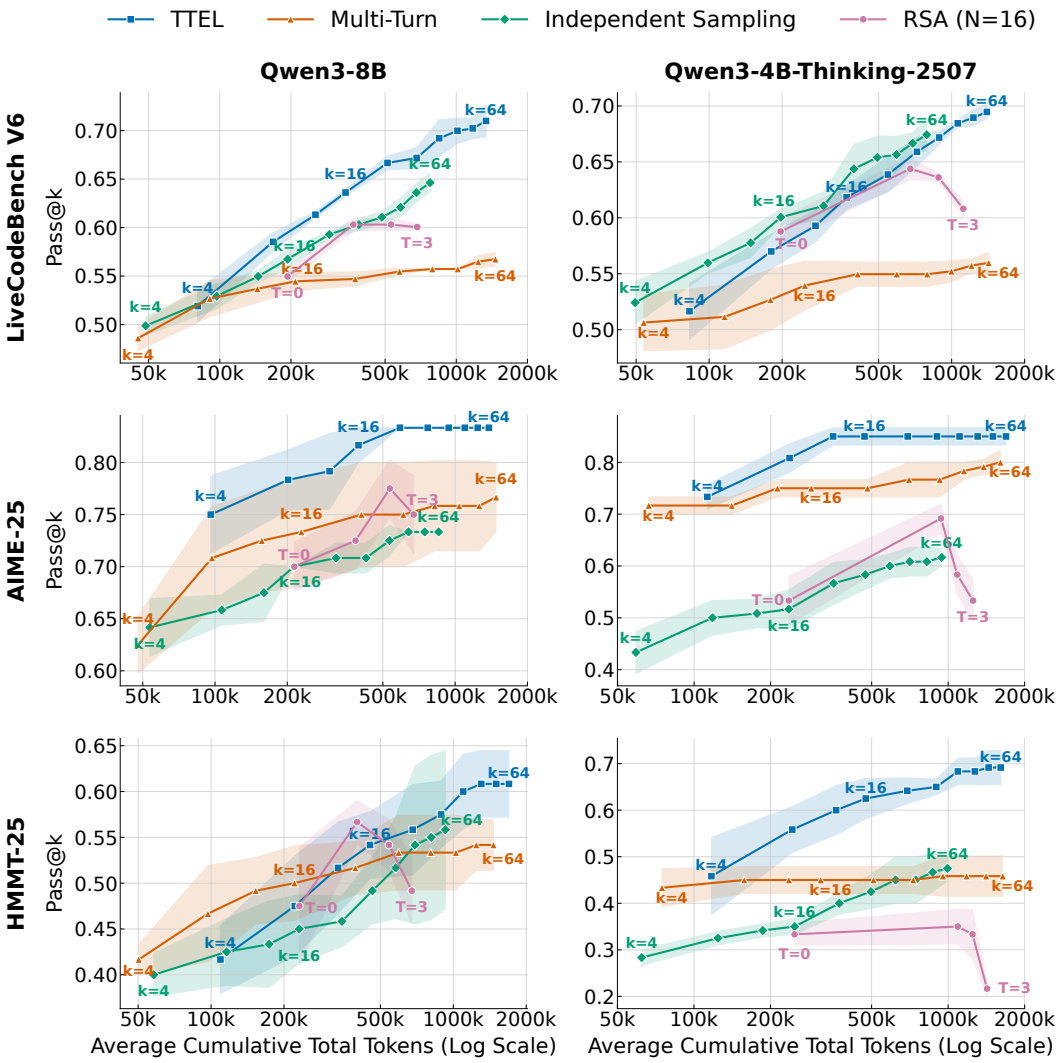


Figure 7: Pareto plot of pass@k vs. average total tokens (input + output) per question on LiveCodeBench V6 (top), AIME-25 (middle) and HMMT-25 (bottom). Shaded regions denote the standard deviation across seeds (3 for LCBv6, 4 for AIME-25 and HMMT-25).

Table 1: Inference scaling and error analysis for Qwen3-8B on LiveCodeBench (LCB). We evaluate pass@k (%) and average generation cost per question over 3 different seeds.

		pass@k (%)				**Generated Tokens** ($\times 10^3$)			
Method	k	Mean $\pm$ SD	s=0	s=1	s=2	Mean $\pm$ SD	s=0	s=1	s=2
TTEL	4	51.9 ± 1.6	54.2	51.1	50.4	27.2 ± 0.5	27.3	26.6	27.7
	8	58.5 ± 1.0	58.8	57.3	59.5	49.7 ± 0.4	50.2	49.3	49.7
	12	61.3 ± 0.4	61.8	61.1	61.1	72.4 ± 0.2	72.6	72.4	72.0
	16	63.6 ± 0.4	63.4	64.1	63.4	95.6 ± 0.2	95.4	95.8	95.6
	24	66.7 ± 0.7	67.2	65.6	67.2	141.7 ± 0.9	140.5	142.7	141.8
	32	67.2 ± 1.1	67.9	65.6	67.9	186.0 ± 0.6	186.4	185.1	186.4
	40	69.2 ± 1.9	71.8	67.2	68.7	230.3 ± 0.3	230.3	230.0	230.6
	48	70.0 ± 1.3	71.8	69.5	68.7	274.1 ± 0.8	274.4	273.1	274.9
	56	70.2 ± 1.1	71.8	69.5	69.5	317.6 ± 0.4	317.8	317.0	317.9
	64	71.0 ± 1.6	73.3	70.2	69.5	360.4 ± 0.7	361.3	360.4	359.6
Independent Sampling	4	49.9 ± 1.0	49.6	48.9	51.1	46.1 ± 0.1	46.1	46.2	46.0
	8	52.9 ± 1.3	54.2	51.1	53.4	92.0 ± 0.3	92.0	92.4	91.8
	12	55.0 ± 0.6	55.0	54.2	55.7	137.9 ± 0.3	137.9	138.2	137.5
	16	56.7 ± 1.0	56.5	55.7	58.0	183.7 ± 0.4	184.0	183.9	183.2
	24	59.3 ± 0.4	58.8	59.5	59.5	275.6 ± 0.4	275.8	276.0	275.0
	32	60.3 ± 0.6	59.5	60.3	61.1	367.7 ± 0.6	368.4	367.8	366.8
	40	61.1 ± 0.6	60.3	61.1	61.8	459.7 ± 0.8	460.8	459.4	458.9
	48	62.1 ± 1.0	61.8	61.1	63.4	551.6 ± 1.0	553.0	551.1	550.8
	56	63.6 ± 1.0	64.9	62.6	63.4	643.5 ± 0.8	644.6	643.1	642.8
	64	64.6 ± 1.0	64.9	63.4	65.6	735.0 ± 1.1	736.5	734.6	734.0
Multi-Turn Refinement	4	48.6 ± 1.3	47.3	50.4	48.1	39.7 ± 0.1	39.8	39.6	39.8
	8	52.7 ± 1.9	52.7	55.0	50.4	69.3 ± 0.6	69.5	68.6	69.9
	12	53.7 ± 1.4	52.7	55.7	52.7	93.9 ± 1.0	94.2	92.7	95.0
	16	54.5 ± 1.0	53.4	55.7	54.2	116.0 ± 1.1	116.2	114.6	117.2
	24	54.7 ± 0.7	54.2	55.7	54.2	155.1 ± 1.1	155.4	153.6	156.3
	32	55.5 ± 0.4	55.0	55.7	55.7	190.6 ± 1.8	190.8	188.3	192.6
	40	55.7 ± 0.0	55.7	55.7	55.7	224.9 ± 2.6	225.6	221.4	227.6
	48	55.7 ± 0.0	55.7	55.7	55.7	257.9 ± 2.7	259.1	254.2	260.6
	56	56.5 ± 0.6	56.5	55.7	57.3	290.7 ± 2.9	291.9	286.7	293.4
	64	56.7 ± 0.7	57.3	55.7	57.3	323.2 ± 2.7	324.8	319.4	325.4
RSA	16 (t_0)	55.0 ± 0.6	54.2	55.7	55.0	183.4 ± 0.3	183.3	183.1	183.8
	16 (t_1)	60.3 ± 0.6	59.5	61.1	60.3	324.4 ± 0.3	324.2	324.2	324.8
	16 (t_2)	60.3 ± 0.6	60.3	59.5	61.1	453.7 ± 1.3	452.1	453.9	455.2
	16 (t_3)	60.1 ± 0.4	60.3	59.5	60.3	570.7 ± 1.8	568.7	570.3	573.1

Table 2: Inference scaling and error analysis for Qwen3-8B on AIME 2025. We evaluate pass@k (%) and average generation cost per question over 4 different seeds.

		pass@k (%)					**Generated Tokens ($\times 10^3$)**				
Method	k	Mean $\pm$ SD	s=0	s=1	s=2	s=3	Mean $\pm$ SD	s=0	s=1	s=2	s=3
TTEL	4	75.0 ± 3.7	76.7	80.0	73.3	70.0	36.6 ± 0.7	35.5	37.6	36.6	36.6
	8	78.3 ± 2.9	80.0	80.0	73.3	80.0	70.0 ± 0.5	69.9	70.4	69.1	70.5
	12	79.2 ± 3.6	80.0	80.0	73.3	83.3	101.5 ± 1.1	99.8	101.6	101.6	102.9
	16	81.7 ± 1.7	83.3	80.0	80.0	83.3	132.4 ± 2.4	129.7	130.5	133.8	135.6
	24	83.3 ± 0.0	83.3	83.3	83.3	83.3	192.4 ± 5.1	189.5	189.1	189.8	201.3
	32	83.3 ± 0.0	83.3	83.3	83.3	83.3	248.2 ± 7.8	246.1	243.2	242.1	261.5
	40	83.3 ± 0.0	83.3	83.3	83.3	83.3	301.3 ± 10.6	306.3	295.2	288.0	315.7
	48	83.3 ± 0.0	83.3	83.3	83.3	83.3	351.2 ± 15.2	361.3	343.8	330.2	369.4
	56	83.3 ± 0.0	83.3	83.3	83.3	83.3	396.3 ± 16.2	406.9	392.2	372.0	414.3
	64	83.3 ± 0.0	83.3	83.3	83.3	83.3	439.5 ± 18.0	452.5	437.0	411.0	457.3
Independent Sampling	4	64.2 ± 2.8	60.0	66.7	63.3	66.7	52.7 ± 0.6	52.5	51.9	53.5	52.8
	8	65.8 ± 1.4	66.7	66.7	63.3	66.7	104.9 ± 0.8	105.2	104.6	106.0	104.0
	12	67.5 ± 2.8	70.0	66.7	63.3	70.0	157.3 ± 0.8	157.2	156.9	158.7	156.6
	16	70.0 ± 0.0	70.0	70.0	70.0	70.0	209.7 ± 0.8	209.8	209.2	211.0	209.0
	24	70.8 ± 1.4	70.0	70.0	73.3	70.0	314.3 ± 1.0	314.4	313.6	315.9	313.4
	32	70.8 ± 1.4	70.0	70.0	73.3	70.0	418.8 ± 1.1	419.0	418.0	420.5	417.8
	40	72.5 ± 1.4	73.3	73.3	73.3	70.0	524.1 ± 0.9	524.5	522.9	525.3	523.6
	48	73.3 ± 0.0	73.3	73.3	73.3	73.3	629.2 ± 0.7	629.7	628.0	629.9	629.2
	56	73.3 ± 0.0	73.3	73.3	73.3	73.3	734.8 ± 0.9	734.6	733.6	736.0	735.2
	64	73.3 ± 0.0	73.3	73.3	73.3	73.3	840.1 ± 1.0	839.3	839.1	841.5	840.5
Multi-Turn Refinement	4	62.5 ± 2.8	63.3	60.0	60.0	66.7	42.5 ± 0.8	42.6	43.4	43.0	41.1
	8	70.8 ± 4.9	73.3	63.3	70.0	76.7	73.4 ± 1.5	72.8	75.9	72.7	72.0
	12	72.5 ± 4.9	73.3	66.7	70.0	80.0	100.4 ± 2.3	101.1	103.8	98.3	98.3
	16	73.3 ± 5.3	76.7	66.7	70.0	80.0	125.7 ± 3.4	127.2	130.4	121.2	124.1
	24	75.0 ± 5.0	80.0	70.0	70.0	80.0	170.5 ± 4.2	172.5	174.8	163.5	171.1
	32	75.0 ± 5.0	80.0	70.0	70.0	80.0	207.4 ± 3.9	206.1	209.8	201.7	211.9
	40	75.8 ± 4.3	80.0	73.3	70.0	80.0	242.6 ± 5.7	237.6	245.6	236.7	250.5
	48	75.8 ± 4.3	80.0	73.3	70.0	80.0	275.8 ± 6.6	269.4	279.0	269.8	285.1
	56	75.8 ± 4.3	80.0	73.3	70.0	80.0	308.8 ± 7.0	301.1	312.5	303.1	318.4
	64	76.7 ± 3.3	80.0	73.3	73.3	80.0	341.9 ± 7.1	333.2	345.5	337.3	351.5
RSA	16 (t_0)	70.0 ± 2.4	73.3	70.0	70.0	66.7	210.3 ± 1.0	211.5	210.9	208.8	210.0
	16 (t_1)	72.5 ± 2.8	73.3	70.0	70.0	76.7	338.2 ± 2.0	341.0	338.6	335.6	337.6
	16 (t_2)	77.5 ± 2.8	80.0	76.7	73.3	80.0	445.6 ± 4.2	452.9	442.6	442.8	444.1
	16 (t_3)	75.0 ± 3.7	80.0	73.3	70.0	76.7	538.4 ± 8.3	552.7	533.4	534.9	532.5

Table 3: Inference scaling and error analysis for Qwen3-8B on HMMT. We evaluate pass@k (%) and average generation cost per question over 4 different seeds.

		pass@k (%)					**Generated Tokens** ($\times 10^3$)				
Method	k	Mean $\pm$ SD	s=0	s=1	s=2	s=3	Mean $\pm$ SD	s=0	s=1	s=2	s=3
TTEL	4	41.7 ± 3.7	40.0	46.7	43.3	36.7	40.9 ± 1.5	42.4	38.4	41.9	41.0
	8	47.5 ± 5.5	43.3	56.7	43.3	46.7	75.1 ± 2.1	76.5	71.4	75.9	76.6
	12	51.7 ± 5.0	50.0	60.0	50.0	46.7	110.2 ± 3.3	109.5	105.2	112.4	113.8
	16	54.2 ± 3.6	53.3	60.0	53.3	50.0	146.6 ± 4.5	142.8	141.5	150.1	152.0
	24	55.8 ± 4.9	56.7	63.3	53.3	50.0	216.7 ± 8.3	207.9	212.2	216.5	230.0
	32	57.5 ± 3.6	56.7	63.3	53.3	56.7	280.7 ± 15.6	265.1	269.7	283.0	305.2
	40	60.0 ± 4.1	56.7	66.7	56.7	60.0	344.6 ± 22.2	322.1	328.4	348.9	379.0
	48	60.8 ± 3.6	56.7	66.7	60.0	60.0	408.2 ± 32.8	379.5	377.4	418.7	457.4
	56	60.8 ± 3.6	56.7	66.7	60.0	60.0	469.5 ± 42.7	434.3	428.4	480.5	534.8
	64	60.8 ± 3.6	56.7	66.7	60.0	60.0	527.9 ± 52.3	490.0	474.9	537.6	609.3
Independent Sampling	4	40.0 ± 2.4	40.0	36.7	43.3	40.0	57.4 ± 0.6	57.1	57.8	56.5	58.0
	8	42.5 ± 3.6	46.7	36.7	43.3	43.3	114.7 ± 0.8	114.7	115.0	113.4	115.6
	12	43.3 ± 4.7	50.0	36.7	43.3	43.3	171.8 ± 0.8	171.4	172.2	170.8	172.8
	16	45.0 ± 3.7	50.0	40.0	43.3	46.7	228.8 ± 1.0	227.5	229.3	228.2	230.2
	24	45.8 ± 2.8	50.0	43.3	43.3	46.7	343.3 ± 1.8	341.1	343.4	342.7	346.0
	32	49.2 ± 3.6	53.3	50.0	43.3	50.0	458.0 ± 1.7	456.2	458.3	456.8	460.7
	40	51.7 ± 5.5	56.7	56.7	43.3	50.0	572.5 ± 2.3	570.1	572.5	571.1	576.1
	48	54.2 ± 8.6	66.7	56.7	43.3	50.0	687.5 ± 2.8	684.6	688.0	685.7	691.8
	56	55.0 ± 9.0	66.7	60.0	43.3	50.0	802.2 ± 2.9	799.4	801.9	800.6	806.9
	64	55.8 ± 8.6	66.7	60.0	43.3	53.3	917.1 ± 2.7	914.5	916.7	915.8	921.7
Multi-Turn Refinement	4	41.7 ± 1.7	43.3	43.3	40.0	40.0	45.6 ± 1.1	46.5	45.1	46.7	44.0
	8	46.7 ± 5.3	43.3	53.3	50.0	40.0	74.2 ± 3.2	77.3	70.6	77.3	71.5
	12	49.2 ± 3.6	50.0	53.3	50.0	43.3	96.9 ± 4.5	101.8	91.1	100.7	94.0
	16	50.0 ± 4.1	50.0	53.3	53.3	43.3	116.4 ± 6.1	122.8	109.1	121.9	111.7
	24	51.7 ± 2.9	53.3	53.3	53.3	46.7	153.1 ± 7.4	160.4	143.9	160.4	147.6
	32	53.3 ± 4.1	56.7	53.3	56.7	46.7	187.3 ± 7.5	193.8	178.3	195.6	181.5
	40	53.3 ± 4.1	56.7	53.3	56.7	46.7	220.6 ± 8.0	226.8	210.4	229.8	215.3
	48	53.3 ± 4.1	56.7	53.3	56.7	46.7	253.7 ± 6.6	259.8	244.8	260.1	250.0
	56	54.2 ± 2.8	56.7	53.3	56.7	50.0	287.5 ± 4.8	292.5	279.9	290.2	287.4
	64	54.2 ± 2.8	56.7	53.3	56.7	50.0	320.9 ± 4.6	326.5	313.8	322.5	320.8
RSA	16 (t_0)	47.5 ± 6.4	43.3	40.0	50.0	56.7	229.0 ± 0.4	229.3	229.2	228.3	229.4
	16 (t_1)	56.7 ± 2.4	56.7	56.7	53.3	60.0	365.0 ± 3.7	358.6	367.2	366.0	368.1
	16 (t_2)	54.2 ± 2.8	56.7	53.3	50.0	56.7	461.8 ± 4.9	453.4	464.2	465.5	464.1
	16 (t_3)	49.2 ± 3.6	50.0	43.3	50.0	53.3	547.5 ± 4.7	539.5	550.9	551.0	548.6

Table 4: Inference scaling and error analysis for Qwen3-4B-Thinking-2507 on LiveCodeBench (LCB). We evaluate pass@k (%) and average generation cost per question over 3 different seeds.

		pass@k (%)				**Generated Tokens** ($\times 10^3$)			
Method	k	Mean $\pm$ SD	s=0	s=1	s=2	Mean $\pm$ SD	s=0	s=1	s=2
TTEL	4	51.7 ± 2.5	53.4	53.4	48.1	25.4 ± 0.8	26.5	25.1	24.7
	8	57.0 ± 1.6	58.8	57.3	55.0	47.3 ± 0.6	46.7	47.1	48.1
	12	59.3 ± 1.4	60.3	60.3	57.3	67.9 ± 1.1	66.7	67.6	69.3
	16	61.8 ± 1.1	63.4	61.1	61.1	87.9 ± 0.8	86.8	88.0	88.8
	24	63.9 ± 1.6	65.6	61.8	64.1	126.6 ± 2.5	123.1	128.3	128.4
	32	65.9 ± 1.3	66.4	64.1	67.2	165.2 ± 4.6	159.2	170.3	166.1
	40	67.2 ± 0.6	67.9	66.4	67.2	203.4 ± 6.4	195.2	210.8	204.1
	48	68.4 ± 0.4	67.9	68.7	68.7	239.9 ± 8.2	229.4	249.3	241.0
	56	69.0 ± 0.7	67.9	69.5	69.5	276.6 ± 9.4	265.0	288.0	276.8
	64	69.5 ± 0.6	68.7	70.2	69.5	312.8 ± 10.7	299.5	325.7	313.2
Independent Sampling	4	52.4 ± 1.9	55.0	50.4	51.9	47.0 ± 0.3	46.7	46.9	47.5
	8	56.0 ± 1.0	57.3	55.0	55.7	93.9 ± 0.5	93.6	93.5	94.5
	12	57.8 ± 1.3	59.5	56.5	57.3	140.8 ± 0.3	140.5	140.8	141.2
	16	60.1 ± 0.7	61.1	59.5	59.5	187.7 ± 0.3	187.4	187.8	188.0
	24	61.1 ± 1.2	62.6	61.1	59.5	281.6 ± 0.2	281.3	281.5	281.9
	32	64.4 ± 2.2	67.2	61.8	64.1	375.2 ± 0.5	374.8	374.9	375.8
	40	65.4 ± 1.9	67.9	63.4	64.9	468.7 ± 0.5	468.5	468.3	469.4
	48	65.6 ± 1.6	67.9	64.1	64.9	562.7 ± 0.3	562.3	562.7	563.2
	56	66.7 ± 1.0	67.9	65.6	66.4	656.1 ± 0.4	655.6	656.4	656.4
	64	67.4 ± 1.4	69.5	66.4	66.4	749.9 ± 0.1	749.9	750.1	749.8
Multi-Turn Refinement	4	50.6 ± 2.5	53.4	51.1	47.3	29.8 ± 0.3	29.3	29.8	30.1
	8	51.1 ± 2.9	54.2	51.9	47.3	49.4 ± 1.1	47.9	50.2	50.2
	12	52.7 ± 2.7	55.0	54.2	48.9	72.0 ± 1.7	69.7	73.5	72.7
	16	53.9 ± 2.2	56.5	54.2	51.1	94.0 ± 1.2	92.3	94.8	94.9
	24	55.0 ± 1.1	56.5	54.2	54.2	132.1 ± 0.7	131.2	132.1	133.0
	32	55.0 ± 1.1	56.5	54.2	54.2	167.1 ± 0.2	166.9	167.0	167.4
	40	55.0 ± 1.1	56.5	54.2	54.2	200.6 ± 0.6	200.4	200.0	201.4
	48	55.2 ± 1.0	56.5	55.0	54.2	233.1 ± 0.2	232.7	233.1	233.3
	56	55.7 ± 0.6	56.5	55.7	55.0	265.6 ± 0.8	264.4	266.3	266.1
	64	56.0 ± 1.0	57.3	55.7	55.0	298.1 ± 1.5	296.0	299.4	299.0
RSA	16 (t_0)	58.8 ± 1.2	58.8	60.3	57.3	187.2 ± 0.2	187.0	187.2	187.5
	16 (t_1)	64.4 ± 1.0	63.4	64.1	65.6	269.0 ± 1.0	270.2	267.7	269.2
	16 (t_2)	63.6 ± 0.4	64.1	63.4	63.4	415.7 ± 0.8	415.3	415.1	416.8
	16 (t_3)	60.8 ± 0.7	60.3	61.8	60.3	522.9 ± 2.3	522.8	520.2	525.7

Table 5: Inference scaling and error analysis for Qwen3-4B-Thinking-2507 on AIME 2025. We evaluate pass@k (%) and average generation cost per question over 4 different seeds.

		pass@k (%)					**Generated Tokens** ($\times 10^3$)				
Method	k	Mean $\pm$ SD	s=0	s=1	s=2	s=3	Mean $\pm$ SD	s=0	s=1	s=2	s=3
TTEL	4	73.3 ± 2.4	76.7	73.3	70.0	73.3	40.3 ± 0.5	39.8	40.4	40.0	41.0
	8	80.8 ± 2.8	83.3	76.7	83.3	80.0	76.7 ± 1.1	77.4	75.6	75.6	78.1
	12	85.0 ± 1.7	86.7	83.3	86.7	83.3	108.2 ± 2.1	109.8	104.7	109.7	108.4
	16	85.0 ± 1.7	86.7	83.3	86.7	83.3	141.9 ± 4.3	144.0	136.5	147.6	139.4
	24	85.0 ± 1.7	86.7	83.3	86.7	83.3	205.4 ± 8.4	206.4	198.9	218.7	197.5
	32	85.0 ± 1.7	86.7	83.3	86.7	83.3	264.7 ± 11.8	259.6	261.7	284.4	253.1
	40	85.0 ± 1.7	86.7	83.3	86.7	83.3	323.6 ± 15.4	311.0	321.7	349.4	312.4
	48	85.0 ± 1.7	86.7	83.3	86.7	83.3	379.4 ± 22.2	353.1	380.9	413.9	369.9
	56	85.0 ± 1.7	86.7	83.3	86.7	83.3	437.0 ± 28.5	404.6	433.2	482.7	427.3
	64	85.0 ± 1.7	86.7	83.3	86.7	83.3	491.0 ± 33.0	451.9	481.6	543.2	487.4
Independent Sampling	4	43.3 ± 4.1	36.7	43.3	46.7	46.7	58.1 ± 0.5	59.0	57.8	57.8	58.0
	8	50.0 ± 3.3	46.7	53.3	46.7	53.3	116.3 ± 0.7	117.1	115.7	116.8	115.7
	12	50.8 ± 2.8	46.7	53.3	50.0	53.3	174.1 ± 0.5	174.9	173.8	174.1	173.7
	16	51.7 ± 3.7	46.7	53.3	50.0	56.7	232.7 ± 0.8	234.0	232.5	232.4	231.7
	24	56.7 ± 4.1	50.0	60.0	56.7	60.0	349.0 ± 0.6	349.8	348.1	349.2	348.7
	32	58.3 ± 2.9	53.3	60.0	60.0	60.0	464.7 ± 0.7	463.6	464.8	465.4	464.8
	40	60.0 ± 2.4	56.7	60.0	63.3	60.0	581.0 ± 1.0	579.8	582.1	580.2	581.7
	48	60.8 ± 2.8	56.7	63.3	63.3	60.0	696.2 ± 1.9	694.4	698.1	694.3	698.0
	56	60.8 ± 2.8	56.7	63.3	63.3	60.0	812.2 ± 1.5	810.8	813.9	810.5	813.5
	64	61.7 ± 1.7	60.0	63.3	63.3	60.0	928.9 ± 2.8	926.9	931.3	925.4	932.0
Multi-Turn Refinement	4	71.7 ± 1.7	73.3	73.3	70.0	70.0	31.2 ± 0.8	32.0	30.6	30.3	32.0
	8	71.7 ± 1.7	73.3	73.3	70.0	70.0	45.8 ± 0.8	46.5	44.7	45.3	46.7
	12	75.0 ± 1.7	76.7	76.7	73.3	73.3	64.1 ± 1.3	65.7	63.5	62.4	65.0
	16	75.0 ± 1.7	76.7	76.7	73.3	73.3	84.0 ± 1.9	85.1	83.1	81.5	86.4
	24	75.0 ± 1.7	76.7	76.7	73.3	73.3	119.0 ± 2.8	120.3	116.4	116.3	122.9
	32	76.7 ± 3.3	80.0	80.0	73.3	73.3	153.2 ± 5.3	154.2	147.4	149.9	161.4
	40	76.7 ± 3.3	80.0	80.0	73.3	73.3	188.3 ± 6.3	189.6	181.2	184.5	197.8
	48	78.3 ± 1.7	80.0	80.0	76.7	76.7	224.4 ± 6.3	227.3	216.4	221.0	232.9
	56	79.2 ± 1.4	80.0	80.0	76.7	80.0	261.1 ± 7.5	265.4	251.8	256.4	271.0
	64	80.0 ± 2.4	83.3	80.0	76.7	80.0	297.8 ± 7.3	302.9	288.4	293.4	306.7
RSA	16 (t_0)	53.3 ± 4.7	53.3	60.0	46.7	53.3	232.6 ± 0.6	232.6	231.8	233.5	232.6
	16 (t_1)	69.2 ± 2.8	66.7	70.0	73.3	66.7	268.4 ± 1.8	270.1	266.4	266.9	270.3
	16 (t_2)	58.3 ± 3.7	63.3	56.7	60.0	53.3	375.7 ± 4.3	378.8	369.2	374.6	380.3
	16 (t_3)	53.3 ± 4.1	56.7	53.3	56.7	46.7	449.2 ± 5.5	446.4	444.7	447.1	458.6

Table 6: Inference scaling and error analysis for Qwen3-4B-Thinking-2507 on HMMT. We evaluate pass@k (%) and average generation cost per question over 4 different seeds.

		pass@k (%)					**Generated Tokens ($\times 10^3$)**				
Method	k	Mean $\pm$ SD	s=0	s=1	s=2	s=3	Mean $\pm$ SD	s=0	s=1	s=2	s=3
TTEL	4	45.8 ± 8.3	53.3	43.3	53.3	33.3	43.3 ± 0.7	42.6	42.6	43.8	44.2
	8	55.8 ± 4.9	56.7	53.3	63.3	50.0	81.5 ± 0.9	80.5	81.3	82.9	81.2
	12	60.0 ± 5.3	63.3	53.3	66.7	56.7	115.4 ± 1.3	116.0	113.2	116.0	116.4
	16	62.5 ± 4.3	66.7	56.7	66.7	60.0	147.9 ± 3.5	151.1	145.4	151.5	143.5
	24	64.2 ± 2.8	66.7	60.0	66.7	63.3	210.7 ± 7.8	217.9	200.9	218.7	205.3
	32	65.0 ± 1.7	66.7	63.3	66.7	63.3	269.6 ± 10.6	279.9	256.2	280.1	262.2
	40	68.3 ± 2.9	70.0	63.3	70.0	70.0	325.1 ± 14.2	338.2	305.8	339.2	317.4
	48	68.3 ± 2.9	70.0	63.3	70.0	70.0	375.6 ± 15.7	390.5	355.9	391.4	364.4
	56	69.2 ± 3.6	70.0	63.3	73.3	70.0	424.4 ± 18.8	438.5	403.4	447.4	408.5
	64	69.2 ± 3.6	70.0	63.3	73.3	70.0	472.5 ± 23.1	487.7	447.5	502.2	452.6
Independent Sampling	4	28.3 ± 1.7	26.7	30.0	30.0	26.7	61.7 ± 0.5	62.1	60.9	62.2	61.6
	8	32.5 ± 1.4	33.3	33.3	33.3	30.0	123.1 ± 0.9	123.6	121.5	123.9	123.5
	12	34.2 ± 1.4	36.7	33.3	33.3	33.3	184.7 ± 1.0	184.8	183.1	185.6	185.3
	16	35.0 ± 1.7	36.7	36.7	33.3	33.3	246.7 ± 1.0	246.6	245.1	247.8	247.1
	24	40.0 ± 2.4	43.3	40.0	40.0	36.7	369.5 ± 1.3	369.4	367.7	371.3	369.5
	32	42.5 ± 3.6	46.7	43.3	43.3	36.7	492.4 ± 0.9	491.2	492.0	493.7	492.7
	40	45.0 ± 5.0	46.7	50.0	46.7	36.7	615.0 ± 0.4	614.4	615.0	615.0	615.6
	48	45.0 ± 5.0	46.7	50.0	46.7	36.7	737.7 ± 0.9	736.6	737.5	737.4	739.2
	56	46.7 ± 4.1	50.0	50.0	46.7	40.0	860.9 ± 0.9	859.7	860.8	860.6	862.3
	64	47.5 ± 2.8	50.0	50.0	46.7	43.3	984.2 ± 1.3	982.8	985.5	982.8	985.5
Multi-Turn Refinement	4	43.3 ± 4.1	40.0	43.3	50.0	40.0	33.9 ± 0.5	33.3	33.9	34.5	34.1
	8	45.0 ± 2.9	43.3	43.3	50.0	43.3	49.1 ± 0.4	48.7	49.0	49.1	49.7
	12	45.0 ± 2.9	43.3	43.3	50.0	43.3	67.5 ± 0.3	68.0	67.6	67.0	67.5
	16	45.0 ± 2.9	43.3	43.3	50.0	43.3	89.7 ± 1.1	88.9	91.7	89.3	88.9
	24	45.0 ± 2.9	43.3	43.3	50.0	43.3	127.7 ± 2.6	127.8	131.9	125.9	125.3
	32	45.0 ± 2.9	43.3	43.3	50.0	43.3	163.8 ± 3.4	162.9	169.0	163.6	159.6
	40	45.8 ± 4.3	43.3	43.3	53.3	43.3	202.1 ± 4.3	199.0	208.1	204.2	197.2
	48	45.8 ± 4.3	43.3	43.3	53.3	43.3	239.6 ± 5.3	233.7	246.8	242.6	235.5
	56	45.8 ± 4.3	43.3	43.3	53.3	43.3	277.7 ± 6.1	270.2	286.3	280.0	274.5
	64	45.8 ± 4.3	43.3	43.3	53.3	43.3	316.3 ± 7.7	306.0	326.9	319.3	313.1
RSA	16 (t_0)	33.3 ± 2.4	33.3	30.0	36.7	33.3	246.4 ± 0.5	246.6	245.6	246.8	246.5
	16 (t_1)	35.0 ± 3.7	40.0	33.3	30.0	36.7	278.5 ± 2.0	280.2	276.1	276.9	280.8
	16 (t_2)	33.3 ± 5.3	30.0	40.0	26.7	36.7	394.9 ± 2.6	396.7	390.5	397.0	395.5
	16 (t_3)	21.7 ± 1.7	23.3	20.0	20.0	23.3	473.7 ± 1.5	475.6	474.0	473.8	471.4

G QUALITATIVE EXAMPLE

$\Delta^{(T)} < 0.2$ $0.2 \leq \Delta^{(T)} < 0.5$ $\Delta^{(T)} \geq 0.5$ **bold** = branch point

But how to model this ? Let's think of the problem as: for each query, the minimal stairs is the minimal sum of vertical steps between the start and end positions, given that we can move via walkways between adjacent buildings (as long as the destination building's floor count is sufficient for the current floor).
But how can we model this?
Another way to think: the minimal number of stairs required is the sum of the vertical steps between the starting floor and the ending floor, minus any possible reductions. Wait , no. Because you can choose different paths to change floors . For example, moving to a higher floor first, then down, etc.
Wait , perhaps the minimal stairs is (Y_i - Z_i) if Y_i > Z_i, but that's not always the case. For example, if there's a way to go up from Y_i to a higher floor, then down to Z_i, but that's adding stairs. So maybe that's not helpful.

(a) Spike Detection: all candidate teacher spikes ($\Delta^{(T)} > 0.06$).

$\Delta^{(B)} < 0.2$ $0.2 \leq \Delta^{(B)} < 0.5$ $\Delta^{(B)} \geq 0.5$

But how to model this ? Let's think of the problem as: for each query, the minimal stairs is the minimal **sum** of vertical steps between the start and end positions, given that we can move via walkways between adjacent buildings (as long as the destination building's floor count is sufficient for the current floor).
But how can we model this?
Another way to think: the minimal number of stairs required is the sum of the vertical steps between the starting floor and the ending floor, minus any possible reductions. Wait , no. Because you can choose different paths to change floors . For example, moving to a higher floor first, then down, etc.
Wait , perhaps the minimal stairs is (Y_i - Z_i) if Y_i > Z_i, but that's not always the case. For example, if there's a way to go up from Y_i to a higher floor, then down to Z_i, but that's adding stairs. So maybe that's not helpful.

(b) Spike Filtering via Spike Baseline. Blue = surviving spikes, Orange = baseline spikes.

But how to model this? Let's think of the problem as: for each query, the minimal stairs is the minimal **number** of stairs required to adjust floors when moving between buildings.
Here's an insight: The minimal number of stairs is the sum of the vertical steps needed to adjust from the starting floor to a certain floor (say, X) in the starting building, then move via walkways to the ending building, and then adjust from X to the ending floor. The total stairs is |Y - X| + |Z - X|. To minimize this, we need to choose X such that there exists a path from the starting building to the ending building with all buildings along the path having at least X floors.
The minimal value of |Y - X| + |Z - X| is |Y - Z|, which is achieved when X is between Y and Z. So, if there exists such an X, the answer is |Y - Z|. Otherwise, we need to choose the best possible X.
Thus, the approach is:

(c) Correction: branching from the selected spike leads to a successful answer (reward = 1.0).

Figure 8: Qualitative analysis of spike detection and correction for Question 21 in LCBv6.

The steps of spike detection, spike filtering and branching are illustrated in fig. 8

a) The teacher student probability difference spikes at many points, colored in blue according to magnitude ($\Delta^{(T)}$).

b) Spike filtering removes tokens where the student-baseline gap ($\Delta^{(B)}$) is also large (colored in orange according to magnitude), isolating only spikes with genuine divergence in reasoning between student and teacher.

c) When the model re-generates from the chosen surviving spike (**sum**), it branches into a correct reasoning path which eventually leads to the correct answer.

H Sensitivity Analysis of Spike Threshold

Table 7: Hyperparameter sweep for Spike Threshold (Qwen3-8B, LCBv6, $k = 16$ rollouts). Mean and standard deviation over 3 seeds.

τ	**pass@k (%)**	**Generated Tokens** ($\times 10^3$)	**Turn 1 Spikes**
0.04	64.4 ± 0.5	93.9 ± 1.1	20.0 ± 0.6
0.06	65.9 ± 1.6	94.3 ± 1.1	17.9 ± 0.4
0.08	64.4 ± 1.6	96.6 ± 0.7	16.7 ± 0.8
0.1	65.4 ± 2.3	95.6 ± 0.9	15.5 ± 0.3
0.12	65.7 ± 1.3	96.7 ± 0.3	14.2 ± 0.4
0.14	64.4 ± 1.9	97.3 ± 1.0	12.9 ± 0.5

Table 8: Hyperparameter sweep for Spike Threshold (Qwen3-8B, LCBv6, $k = 16$ rollouts). Results per seed (denoted by s)).

	pass@k (%)			**Generated Tokens** ($\times 10^3$)			**Turn 1 Spikes**		
τ	s=0	s=1	s=2	s=0	s=1	s=2	s=0	s=1	s=2
0.04	64.9	64.1	64.1	93.0	93.7	95.1	20.3	20.6	19.3
0.06	67.2	64.1	66.4	94.6	93.1	95.2	18.3	17.8	17.6
0.08	64.9	65.6	62.6	95.8	97.0	96.9	17.4	15.8	16.9
0.1	67.9	63.4	64.9	94.7	95.5	96.5	15.8	15.1	15.5
0.12	66.4	66.4	64.1	96.4	96.9	96.9	14.7	14.0	13.9
0.14	64.1	66.4	62.6	96.9	96.6	98.5	13.4	12.9	12.4

I BRANCHING ANALYSIS

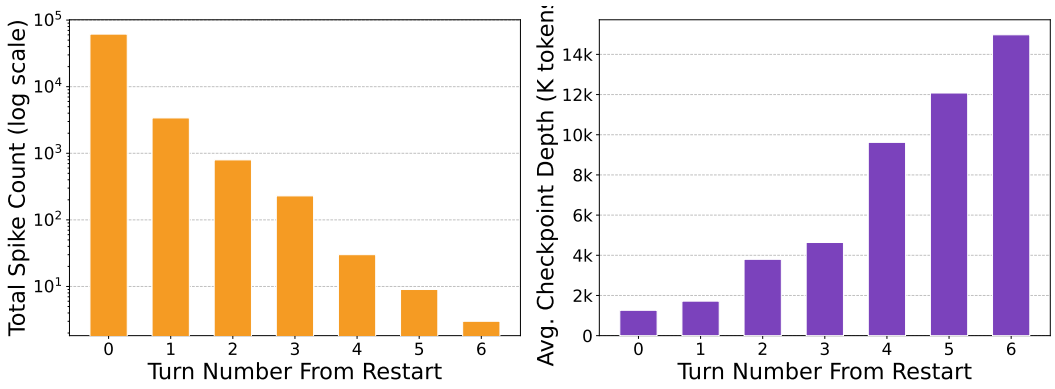


Figure 9: Total spike count and average checkpoint depth for successive turn numbers (relative to restarts), evaluated on LiveCodeBench V6 with Qwen3-8B ($k = 16$) using TTEL-GenFB. The number of spikes decreases between successive turns.

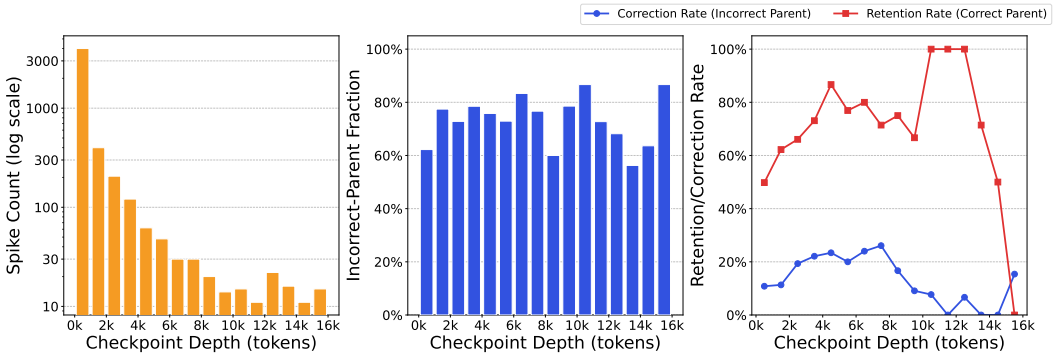


Figure 10: Spike count, fraction of spikes with an incorrect parent, and success rates based on Checkpoint Depth collected for TTEL-GenFB on LiveCodeBench V6 with Qwen3-8B ($k = 16$). A spike has a correct parent if the answer from that turn is already correct, and an incorrect parent otherwise. We define the success rate of the branched rollout as the correction rate for spikes with an incorrect parent and the retention rate for spikes with a correct parent. We observe that retention rate is greater than correction rate. Correction rate decreases and retention rate increases after a checkpoint depth of 8k tokens.