Adaptive Generate-Rank-Verify: Inference-Time Search with Costly Verification

Shaddin Dughmi[⋆1] **Mahdi Haghifam**[⋆2] **Yusuf Hakan Kalayci**[⋆3]
[1]University of Southern California
[2]Toyota Technological Institute at Chicago
[3]IDEAL Institute, University of Illinois Chicago
[⋆]Authors listed in alphabetical order

Abstract

Many inference-time language-model pipelines combine a cheap reward signal with an expensive verifier, such as exact answer checking in mathematical reasoning or hidden-test execution in code generation. We formalize this setting using a learning-theoretic lens as *generative active search*: a cost-sensitive first-positive search problem in which a policy adaptively samples candidates from an unknown distribution, observes cheap scores, and pays for verifier labels until it finds a positive example. For a fixed prompt, the generator and reward model induce two unknown objects: a distribution over reward scores and a score-conditioned success function. When these quantities are known, we characterize the distribution-aware optimal policy using a dynamic programming approach. In the realistic and practical setting where both the score distribution and success function are unknown, we propose ADAP, a shellwise adaptive generate-rank-verify algorithm that progressively increases the number of sampled responses and top-ranked verifications. Under the monotonicity assumption that higher reward scores are no less likely to pass verification, we show that ADAP achieves expected cost within a constant factor of the distribution-aware optimum. We complement this result with learning-theoretic lower bounds, based on a centered star number, showing that structural assumptions on the score–label relationship are necessary. Experiments on mathematical reasoning and competitive programming validate the predicted advantage over both fixed non-adaptive policies and difficulty-adaptive baselines.

1 Introduction

State-of-the-art large language models (LLMs) increasingly rely on *inference-time search*: they generate multiple candidate solutions, use a cheap but noisy reward signal to triage them, and reserve an expensive verifier for the few candidates that may be correct. This paradigm underlies recent progress on difficult reasoning tasks, including competitive programming and mathematical Olympiad exams [Goo25; Ope24], where verification may take the form of hidden-test execution, exact answer checking, or proof verification. Because each additional generation, reward score, and verifier call incurs *cost*—in wall-clock latency, GPU compute, or API spend—these systems face a basic compute-allocation problem at inference time. In a nutshell: given an LLM, a cheap but noisy reward model, and a costly verifier, how should one use reward scores to decide when to keep sampling and when to spend verification calls, in order to find a verified-correct response with minimal total cost? This question has motivated a growing line of work on test-time compute allocation and adaptive inference-time search [Sne+25; Dam+25; Zha+26; RAK25; KRD25; Wan+25b; Qu26].

The challenge is not merely that inference-time compute is costly; it is that the right allocation is inherently prompt-dependent. A natural non-adaptive baseline fixes a pair $(N_{\text{rew}}, N_{\text{ver}})$: generate N_{rew} candidate responses, rank them by reward score, and verify the top N_{ver}. However, no single choice of $(N_{\text{rew}}, N_{\text{ver}})$ is appropriate across all prompts. As shown in Fig. 1, the per-prompt optimal choice

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

varies widely. Easy instances may require only a small candidate pool and one verification call, while hard instances may require orders of magnitude more generation and several verification attempts. Similar prompt-dependent variation in the value of test-time compute has also been observed in recent studies [Sne+25; Dam+25; Zha+26; RAK25; HMZ26]. Thus, any fixed policy must either overspend on easy prompts or under-search on hard ones.

One natural response is to learn a predictor of per-prompt difficulty from historical data and use it to choose the inference-time budget [Sne+25; Dam+25]. Such approaches can be effective when the deployment prompt distribution is stable and the learned predictor remains well calibrated. However, this assumption can fail under distribution shift: the mix of prompts may change over time, and the relationship between reward scores and verifier acceptance may vary across prompts or domains. In such settings, historical tuning or a single global calibration rule may not transfer reliably. This motivates online, prompt-dependent policies that adapt using only the reward scores and verification outcomes observed on the current prompt.

These observations lead to the central question we aim to answer in this paper: Can an online policy use an uncalibrated reward model, adapt to prompt difficulty, and achieve near-optimal cost relative to a distribution-aware policy that knows the score distribution and reward–verifier relationship? We answer this question by formulating generate–rank–verify inference as a cost-sensitive search problem and by developing an adaptive algorithm with provable and empirical guarantees across various cost regimes.

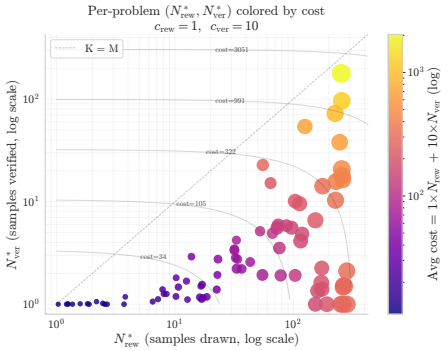


Figure 1: Per-problem optimal non-adaptive choices $(\overline{N^\star_{\text{rew}}}, \overline{N^\star_{\text{ver}}})$ on LiveCodeBench (100 permutations each). For each problem, $N^\star_{\text{rew}}$ is the number of generated candidates and $N^\star_{\text{ver}}$ is the number of top-ranked candidates verified. Color encodes total cost; faint diagonals are iso-cost contours. The wide spread shows that fixed $(N_{\text{rew}}, N_{\text{ver}})$ policies can overspend on easy problems and under-spend on hard ones.

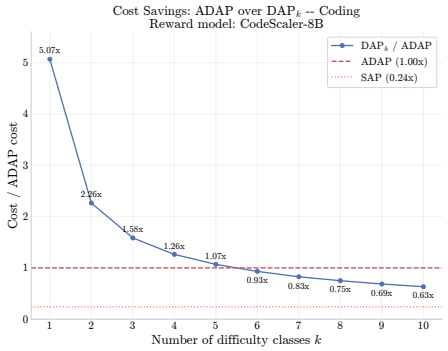


Figure 2: The plot shows average cost ratios of two baselines—Difficulty Adaptive Policy (DAP$_k$) and SAMPLEAWARE Policy—relative to ADAP. SAMPLEAWARE retroactively picks the minimum-cost generation count N_{rew} and verification count N_{ver} such that verifying the N_{ver} highest-reward samples among N_{rew} finds a correct answer. DAP$_k$ estimates problem difficulty from the fraction of successful samples, partitions them into k classes optimally, and assigns each the cheapest uniform strategy ensuring 100% success.

1.1 Contributions

We study inference-time generate–rank–verify pipelines, where candidate responses are sampled from a language model, scored by a cheap reward model, and checked by an expensive verifier. Motivated by the compute-allocation question above, we formalize this setting as a cost-sensitive sequential decision problem, which we call *active search*. Let $c_{\text{rew}} > 0$ denote the cost of generating one candidate response and scoring it with the reward model, and let $c_{\text{ver}} > 0$ denote the cost of one verifier call. For each prompt x, the generator and reward model induce an unknown distribution $\mathcal{D}_x \in \Delta(\mathbb{R})$ over reward scores. The verifier induces an unknown score-conditioned success function $h^\star_x : \mathbb{R} \to [0, 1]$, where $h^\star_x(r)$ is the probability that a candidate with reward score r passes verification. A sound policy adaptively generates candidates, observes their reward scores, and chooses which candidates to verify. It pays $J = c_{\text{rew}} N_{\text{rew}} + c_{\text{ver}} N_{\text{ver}}$, where N_{rew} is the number of generated-and-scored candidates and N_{ver} is the number of verifier calls, and it may stop only after observing a positive verifier label. This formulation captures the central tension from the introduction: the policy must decide, online and prompt-by-prompt, when to spend cost on more reward-scored samples and when to spend cost on verification.

1. **Optimal Distribution-Aware Benchmark.** We first study the idealized setting in which the prompt-specific score distribution and reward–verifier relationship are known. This gives the benchmark for online adaptation and clarifies the ideal compute-allocation rule. We show that the optimal policy has a simple threshold form: it verifies a candidate exactly when its probability of passing the verifier is high enough to justify the verification cost (Theorem 3.2).

2. **A prompt-dependent online policy: ADAP.** In practice, the policy does not know $\mathcal{D}_x$ or $h_x^\star$, and therefore cannot implement the distribution-aware policy directly. We propose ADAP, a shellwise online algorithm that searches over *dyadic* generation and verification scales (Algorithm 1). ADAP progressively enlarges the candidate pool and verifies top-ranked candidates, using the reward model only as a ranking signal. Under the natural monotonicity assumption that $h_x^\star$ is non-decreasing in the reward score, we prove that ADAP achieves expected cost within a constant factor of the distribution-aware optimum, uniformly over all feasible $(\mathcal{D}_x, h_x^\star)$ (Theorem 4.2).

3. **Learning-theoretic foundations.** The guarantee for ADAP relies on the assumption that higher reward scores are more likely to pass verification. We show that some such inductive bias is unavoidable. Without structural assumptions on the reward–verifier relationship, any sound online policy can be forced to spend much more than the distribution-aware benchmark. We formalize this obstruction through the *centered star number*, a learning-theoretic complexity measure for active search. For binary concept classes, we prove matching lower and upper bounds: the worst-case adaptivity gap is characterized, up to constants, by $\min\{\mathfrak{s}_0, c_{\text{ver}}/c_{\text{rew}}\}$ where $\mathfrak{s}_0$ is the centered star number. We also show a separation between active search and active learning [BBL06; BU16; HY15].

4. **Empirical validation.** We evaluate ADAP on HMMT mathematical reasoning and LiveCodeBench competitive programming tasks, using exact answer matching and hidden-test execution as verifiers. On the feasible subset of prompts with at least one correct sampled candidate, ADAP matches the 100% success rate of the cheapest fixed $(N_{\text{rew}}, N_{\text{ver}})$ policy that also achieves 100% success, while using substantially lower mean cost: $2.9\times$ lower on HMMT and $5.5\times$ lower on LiveCodeBench in our current experiments. At the same mean cost as ADAP, the best fixed policy succeeds on only 74.4% and 88% of trials, respectively. Fig. 2 previews this comparison: ADAP remains competitive with DAP$_k$, an oracle difficulty-stratified proxy for learning-based budget allocation, without using historical prompt-difficulty labels; see Section 5 for details.

2 Problem Setup

Let $\mathcal{X}$ denote the prompt space and $\mathcal{V}$ the vocabulary. The response space $\mathcal{Y} = \bigcup_{t=0}^{\infty} \mathcal{V}^t$ is the set of all sequences of arbitrary length. For each prompt $x \in \mathcal{X}$, the language model induces a distribution $\pi(\cdot \mid x) \in \Delta(\mathcal{Y})$ over candidate responses. We write $Y \sim \pi(\cdot \mid x)$ for one sampled response.

We assume access to two *evaluators* for a prompt-response pair. The first is a reward model $\mathsf{RM} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, which returns a real-valued score. The second is a verifier $\mathsf{Ver} : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$, which returns the final correctness label. We say a response y is *accepted for prompt* x iff $\mathsf{Ver}(x, y) = 1$. Crucially, the reward model is only a proxy for the verifier, and the reward model may be noisy or miscalibrated, and correctness is defined by the verifier.

For a fixed prompt x, the generator and evaluators induce a joint distribution over reward scores and verifier labels. In particular, if $Y \sim \pi(\cdot \mid x)$, define random variables $R_x = \mathsf{RM}(x, Y)$ and $V_x = \mathsf{Ver}(x, Y)$. Then (R_x, V_x) is a random element of $\mathbb{R} \times \{0, 1\}$. We assume throughout that the prompt is feasible, meaning $\Pr_{Y \sim \pi(\cdot|x)}(\mathsf{Ver}(x, Y) = 1) > 0$ since otherwise no generate-and-verify procedure can succeed. The key quantity connecting the reward model to verification is the conditional success probability $h_x^\star(r) = \Pr(V_x = 1 \mid R_x = r)$. Equivalently, $h_x^\star(r)$ is the probability that a candidate with reward score r will pass the verifier. When the prompt is clear from context, we suppress the subscript x and write (R, V) and $h^\star$.

2.1 Active Search

The goal of active search is to find a verified response while minimizing the total cost of generation, reward querying, and verification. The policy sees neither which candidates the verifier will accept nor how reward scores translate into acceptance probabilities; it must learn enough about both, on the fly, to decide when to spend a verification call and when to keep sampling. We formalize the problem

in two steps: Definition 2.1 specifies the problem instance induced by a prompt, and Definition 2.2 specifies what it means for a policy to solve it.

Definition 2.1. Fix a prompt $x \in \mathcal{X}$. The generator $\pi(\cdot \mid x)$, reward model RM, and verifier Ver induce an *active search instance* $(\mathcal{D}_x, h_x^\star)$, where

- $\mathcal{D}_x \in \Delta(\mathbb{R})$ is the marginal distribution of the reward score $R_x = \mathsf{RM}(x, Y)$ when $Y \sim \pi(\cdot \mid x)$;
- $h_x^\star : \mathbb{R} \to [0,1]$ is the score-conditioned success function $h_x^\star(r) = \Pr(V_x = 1 \mid R_x = r)$, where $V_x = \mathsf{Ver}(x, Y)$ when $Y \sim \pi(\cdot \mid x)$. We also assume that $\Pr(V_x = 1) > 0$.

In practice, $\mathcal{D}_x$ is unknown and only accessible via sampling. Furthermore, the true probability that a candidate with score r passes verification, denoted $h_x^\star(r) = \Pr(V_x = 1 \mid R_x = r)$, is also unknown. Therefore, active search is governed by these two unknown objects: the score distribution $\mathcal{D}_x$ and the conditional success function $h_x^\star$. Similar to PAC learning, we impose no assumptions on $\mathcal{D}_x$. However, we assume $h_x \in \mathcal{H}$ for a *probabilistic concept class* $\mathcal{H} \subseteq [0,1]^{\mathbb{R}}$ [KS94]. This is not a strict calibration requirement on the reward model, but rather a necessary inductive bias that allows active search to succeed without exhaustively sampling the space (see Section 6).

Definition 2.2. Fix a probabilistic concept class $\mathcal{H} \subseteq [0,1]^{\mathbb{R}}$. Fix costs $c_{\text{rew}} > 0$ (generation+reward cost per sample) and $c_{\text{ver}} > 0$ (verification cost per sample). A *sound active search policy* is a randomized sequential procedure $\mathcal{A}$ that knows $\mathcal{H}, c_{\text{rew}}, c_{\text{ver}}$ and interacts with an unknown pair $(\mathcal{D}, h^\star)$, where $\mathcal{D} \in \Delta(\mathbb{R})$ and $h^\star \in \mathcal{H}$. The policy maintains a pool of generated but unverified candidates, represented by pairs (Y_i, R_i), where Y_i is the response and R_i is its reward score. At each decision time, based on its history, $\mathcal{A}$ chooses one of two elementary actions:

1. **Generate.** Draw one fresh response $Y \sim \pi(\cdot \mid x)$, observe its reward score $R = \mathsf{RM}(x, Y)$, pay cost c_{rew}, and add (Y, R) to the unverified pool.
2. **Verify.** Choose one pair (Y_i, R_i) from the unverified pool, query its verifier label, pay cost c_{ver}, and remove the pair from the pool. Conditional on R_i, the revealed label has distribution $V_i \sim \text{Bernoulli}(h^\star(R_i))$. If $V_i = 1$, the policy stops and outputs Y_i.

Let N_{rew} be the total number of generated-and-scored candidates, and let N_{ver} be the total number of verifier calls made before stopping. The random total cost is

$$J(\mathcal{A}; h^\star, \mathcal{D}) := c_{\text{rew}} N_{\text{rew}} + c_{\text{ver}} N_{\text{ver}}.$$

If $\mathcal{A}$ never observes a positive verifier label, $J(\mathcal{A}; h^\star, \mathcal{D}) = +\infty$. Finally note that the policy's decisions are measurable with respect to the reward scores of unverified candidates, previously chosen verification indices, and observed verifier labels; it does not inspect the contents of unverified responses except through their reward scores.

3 Distribution-Aware Benchmark

Before tackling the realistic setting where $\mathcal{D}$ and $h^\star$ are unknown, we first establish the fundamental limits of active search by analyzing an idealized algorithm with perfect knowledge of the environment. This distribution-aware optimum provides a strict lower bound on the expected cost and reveals a structural property of the optimal policy that will guide the design of our adaptive algorithm.

Definition 3.1. A *distribution-aware active search policy* is any valid active search policy in the sense of Definition 2.2 that is additionally given perfect knowledge of $(\mathcal{D}, h^\star)$. Let $\Pi_{\text{DA}}(\mathcal{D}, h^\star)$ denote the class of all such policies. The optimal distribution-aware expected cost is

$$J^\star(h^\star, \mathcal{D}) := \inf_{\pi \in \Pi_{\text{DA}}(\mathcal{D}, h^\star)} \mathbb{E}\left[c_{\text{rew}}\, N_{\text{rew}}(\pi) + c_{\text{ver}}\, N_{\text{ver}}(\pi)\right],$$

where policies that never output a verified positive have cost $+\infty$.

We then present the main result whose proof can be found in Section C as Lemmas C.1 and C.2.

Theorem 3.2. *Fix a prompt x, and suppress the subscript x. Let $R \sim \mathcal{D}$, and let $h^\star(r) = \Pr(V = 1 \mid R = r)$. Assume $\Pr(V = 1) > 0$. Define $\tau^\star \in (0,1)$ as the unique solution of*

$$c_{ver}\, \mathbb{E}_{R \sim \mathcal{D}}[\max\{h^\star(R) - \tau^\star, 0\}] = \tau^\star\, c_{rew}.$$

Then the optimal expected cost with knowledge of $(h^\star, \mathcal{D})$ is $J^\star(h^\star, \mathcal{D}) = c_{ver}/\tau^\star$. Moreover, an optimal fully-sequential policy is as follows: after generating a candidate with reward score r, it verifies immediately if $h^\star(r) > \tau^\star$, discards it if $h^\star(r) < \tau^\star$, and breaks ties arbitrarily if $h^\star(r) = \tau^\star$.

Theorem 3.2 shows that the optimal distribution-aware policy is a simple threshold rule: verify a candidate if and only if its conditional success probability exceeds a single break-even threshold. The break-even threshold is based on comparing the immediate chance of success and the continuation value of discarding it and drawing again.

4 ADAP: Adaptive Policy

Theorem 3.2 reveals the structural property of the optimal distribution-aware policy. However, in practice, the true functions governing the environment are unknown. To bridge this gap, we introduce ADAP, an adaptive policy that searches for the right threshold scale online. The policy does not require prior knowledge of $\mathcal{D}_x$ or $h^\star_x$; it only relies on the natural monotonicity assumption that higher reward scores are more likely to correspond to correct outputs. In Section B.2, we empirically validate this assumption on both coding and math benchmarks, both in aggregate and at the per-prompt level.

Assumption 4.1. *We assume $h^\star \in \mathcal{H}_{non\text{-}dec}$ where $\mathcal{H}_{non\text{-}dec} = \{h : \mathbb{R} \to [0,1] \mid \forall\, r_1 \le r_2\ h(r_1) \le h(r_2)\}$ is the family of non-decreasing functions.*

This monotonicity serves as our primary inductive bias, ensuring that the reward model's rankings are informative. As we will formally prove in Section 6, such structural assumptions are not technical conveniences, but fundamental requirements. Now, we are ready to formally state the performance guarantee of ADAP. After that we give an overview of the proof (Formal proof given in Section D)

Algorithm 1 $\mathcal{A}_{\text{ADAP}}$: Shellwise Active Search

Input: costs $c_{\text{ver}}, c_{\text{rew}} > 0$ and prompt x
Output: A response y with observed label $\mathsf{Ver}(x, y) = 1$

$c_{\min} \leftarrow \min\{c_{\text{rew}}, c_{\text{ver}}\}$
$\mathcal{P} \leftarrow \emptyset$ ▷ pool of generated but unverified pairs
for $s = 0, 1, 2, \ldots$ **do**
  $\mathcal{S}_s \leftarrow \left\{(a,b) \in \mathbb{Z}^2_{\ge 0} : a \le b,\ 2^s c_{\min} \le c_{\text{rew}}\, 2^b + c_{\text{ver}}\, 2^{\,b-a} < 2^{s+1} c_{\min}\right\}$
  if $\mathcal{S}_s = \emptyset$ **then**
    continue
  $b^\star_s \leftarrow \max\{b : (a,b) \in \mathcal{S}_s\}$, $j^\star_s \leftarrow \max\{b - a : (a,b) \in \mathcal{S}_s\}$
  $m_s \leftarrow 2^{b^\star_s + 1}$, $k_s \leftarrow 6 \cdot 2^{j^\star_s}$
  Draw fresh responses $y_1, \ldots, y_{m_s} \overset{\text{i.i.d.}}{\sim} \pi(\cdot \mid x)$
  Compute reward scores $\{R_i = \mathsf{RM}(x, y_i)\}_{i \in [m_s]}$ and add $\mathcal{P} \leftarrow \mathcal{P} \cup \{(y_i, R_i) : i \in [m_s]\}$
  Relabel the elements of $\mathcal{P}$ as $(\tilde{y}_1, \tilde{R}_1), \ldots, (\tilde{y}_{|\mathcal{P}|}, \tilde{R}_{|\mathcal{P}|})$ so that $\tilde{R}_1 \ge \tilde{R}_2 \ge \cdots \ge \tilde{R}_{|\mathcal{P}|}$.
  for $j = 1, \ldots, \min\{k_s, |\mathcal{P}|\}$ **do**
    Query $\mathsf{Ver}(x, \tilde{y}_{(j)})$
    Remove $(\tilde{y}_j, \tilde{R}_j)$ from $\mathcal{P}$
    if $\mathsf{Ver}(x, \tilde{y}_{(j)}) = 1$ **then**
      return $\tilde{y}_{(j)}$

Theorem 4.2. *Under the setup of Section 2, fix costs $c_{rew}, c_{ver} > 0$. Let $\mathcal{A}_{\text{ADAP}}$ be the shellwise active search policy in Algorithm 1. Then, for every feasible prompt x, every generator $\pi(\cdot \mid x)$, and every induced pair $(\mathcal{D}, h^\star)$ with $h^\star \in \mathcal{H}_{\text{non-dec}}$ (see Definition 2.1), $\mathcal{A}_{\text{ADAP}}$ is sound and satisfies*

$$\mathbb{E}[J(\mathcal{A}_{\text{ADAP}}; h^\star, \mathcal{D})] \le 154\, J^\star(h^\star, \mathcal{D}),$$

where $J^\star(h^\star, \mathcal{D})$ is the optimal cost of the distribution-aware policy.

Remark 4.3. The multiplicative nature of this guarantee is a critical feature of ADAP. Because the policy's cost scales proportionally with the distribution-aware optimum, "easy" prompts that admit a low optimal cost are answered cheaply. This directly operationalizes our core motivation that test-time compute varies significantly based on the difficulty of the specific prompt (see Fig. 1). ◁

4.1 Main ideas behind ADAP

Fix a prompt x. We will suppress the dependence of parameters on subscript x for brevity. The distribution-aware policy from Section 3 shows that, when $h^\star$ is known and non-decreasing, the right behavior is a threshold rule on the reward score (see Lemma D.1). This motivates comparing ADAP to an arbitrary threshold t. Define

$$q_t = \Pr_{R\sim\mathcal{D}}(R \geq t) \quad \text{and} \quad s_t = \Pr(R \geq t, V = 1) = \mathbb{E}_{R\sim\mathcal{D}}[h^\star(R)\mathbf{1}\{R \geq t\}]\,.$$

A threshold policy that generates candidates and verifies only when $R \geq t$ has expected cost

$$J_t = \frac{c_{\text{rew}} + c_{\text{ver}} q_t}{s_t}.$$

Indeed, each generated candidate costs c_{rew}, incurs an additional verification cost with probability q_t, and succeeds with probability s_t. The unknown quantities q_t and s_t determine the relevant sample and verification scales. If

$$q_t \approx 2^{-a}, \qquad s_t \approx 2^{-b}, \qquad a \leq b,$$

then 2^b generated candidates have constant total success probability above threshold, while the expected number of candidates above threshold is $2^b q_t \approx 2^{b-a}$. Thus the pair (a, b) suggests the natural scales $m \asymp 2^b$ and $k \asymp 2^{b-a}$. The corresponding cost scale is $B_{a,b} = c_{\text{rew}} 2^b + c_{\text{ver}} 2^{b-a}$.

Since ADAP does not know which threshold, and hence which pair (a, b), is appropriate, it searches over these dyadic possibilities by cost scale, similar to the general techniques in [Ces+97]. Shell s contains all pairs (a, b) whose cost $B_{a,b}$ is within a factor of two of $2^s c_{\min}$ (see Lemma D.2). At shell s, the policy chooses the largest sample scale and the largest verification scale represented in that shell, generating m_s new candidates and allowing k_s verifier calls. Starting from small shells and moving geometrically upward ensures that the cost paid before reaching the right scale is only a constant-factor geometric prefix.

The implementation is pool-based. ADAP keeps a pool of generated but unverified responses. Each shell adds its newly generated candidates to this pool, ranks all candidates by reward, and verifies the top k_s. Every queried response is removed from the pool, regardless of the verifier outcome. Thus the pool always contains exactly the candidates whose labels are still unknown, and high-reward candidates discovered in earlier shells remain available to later shells if they were not yet queried.

The monotonicity assumption is what makes this ranking meaningful. Since $h^\star$ is non-decreasing, higher reward scores have weakly larger conditional probability of passing verification. Therefore, at any point, the best use of a limited verification budget is to spend it on the highest-reward unverified candidates. The proof below formalizes this intuition through a success-mass argument: at the correct dyadic scale, the pool's top-ranked candidates contain enough conditional success probability for the shell to succeed with constant probability, and later shells amplify this success probability rapidly.

5 Experiments

5.1 Setup

We evaluate ADAP[1] on two domains where verification is costly relative to sampling: mathematical reasoning and competitive programming.

Mathematical reasoning. We use the HMMT [HMM25] February 2024 and 2025 contests (60 problems total). Solutions are sampled from `Qwen2.5-Math-7B` (base) [Qwe24] at temperature 0.7, top-p 0.95, with $N = 512$ samples per problem. Each candidate is scored by `Qwen2.5-Math-PRM-7B` [Yan+24]; we use the last-step score as the per-sample reward. Verification is exact answer-string matching against the published numerical answer indicated as `boxed{}`. After filtering to problems with at least one correct sample, 27 problems remain. Results with two alternative generators (`Qwen2.5-14B` [Qwe24] and `DeepSeek-R1-Distill-Qwen-7B` [Dee25]) appear in Appendix B.4.1.

Coding. We use medium- and hard-difficulty problems from LiveCodeBench (LeetCode, AtCoder, Codeforces). Solutions are sampled from `Qwen2.5-Coder-3B` (base) [Hui+24] under identical

[1]The implementation of ADAP is publicly available at https://github.com/yhkalayci/efficient_query

settings. Each candidate is scored by `CodeScaler-8B` [Zhu+26]. Verification runs the candidate against the full hidden test suite in a subprocess; a sample is correct only if every test passes. After excluding problems without any correct solution, 83 problems remain.

Cost model and trial protocol. We set $c_{\text{rew}} = 1$ and $c_{\text{ver}} = 10$, so the cost of N_{rew} draws and N_{ver} verifications is $N_{\text{rew}} \cdot c_{\text{rew}} + N_{\text{ver}} \cdot c_{\text{ver}}$. A trial succeeds if at least one verified sample is correct. Each (problem, policy) pair is evaluated over 10 random permutations of sample order, with the same permutation used across policies to enable paired comparison. In Appendix B.2, we demonstrated that the reward model reliably ranks correct samples above incorrect ones with higher probability as we assumed in our theoretical model. To demonstrate cost-ratio robustness across $c_{\text{ver}}/c_{\text{rew}} \in \{1, 10, 20, 30\}$, we presented results in Appendix B.4.2.

5.2 Baselines

Sample-aware lower bound SAMPLEAWARE. For each individual trial, SAMPLEAWARE retroactively selects the cheapest $(N_{\text{rew}}, N_{\text{ver}})$ such that drawing N_{rew} samples and verifying the top-N_{ver} by reward would yield at least one correct answer on that exact trial. The pool and ordering are fixed within a trial, so success is deterministic. SAMPLEAWARE selects budgets, not candidates: if j incorrect candidates outrank the first correct one in the drawn prefix, success requires at least $j + 1$ verifications. Its cost lower-bounds successful fixed-pair policies on that trial, but choosing its budgets requires hindsight verifier labels and is not realizable. We report the mean of these per-trial minimum costs over problems and the 10 permutations.

Difficulty-adaptive policy DAP$_k$. DAP$_k$ exploits knowledge of each problem's difficulty, measured by the empirical pass rate $\hat{r}_p$ defined as number of correct generations divided by total generations. Problems are sorted by $\hat{r}_p$ and partitioned into k contiguous difficulty classes $G_1, \ldots, G_k$; within class G_i the policy commits to a class-specific fixed pair $((N_{\text{rew}})_i, (N_{\text{ver}})_i)$ that achieves 100% success on every problem in G_i at minimum average cost. The partition and all $(((N_{\text{rew}})_i^\star, (N_{\text{ver}})_i^\star))$ are selected jointly by dynamic programming to minimize total mean cost separately for each ordering. DAP$_k$ is an oracle baseline which requires knowing per-problem pass rates in advance. However, it models the difficulty-aware strategies commonly used in practice.

The degenerate case $k = P$, where P is the number of problems, renders DAP$_P$ equivalent to SAMPLEAWARE. In Appendix B.3, we introduce a weaker variant, DAP$_k^{\text{robust}}$, which requires a single partition and one budget pair per class that ensure success across all 10 permutations. In the special case $k = 1$, DAP$_1^{\text{robust}}$ applies the same pair $(N_{\text{rew}}, N_{\text{ver}})$ uniformly to all problems. ADAP is more competitive against this variant: at the default cost ratio, it is cheaper for every reported k on coding and for $k \in \{1, 2, 3, 5\}$ on math. Both variants use verifier labels and are not implementable.

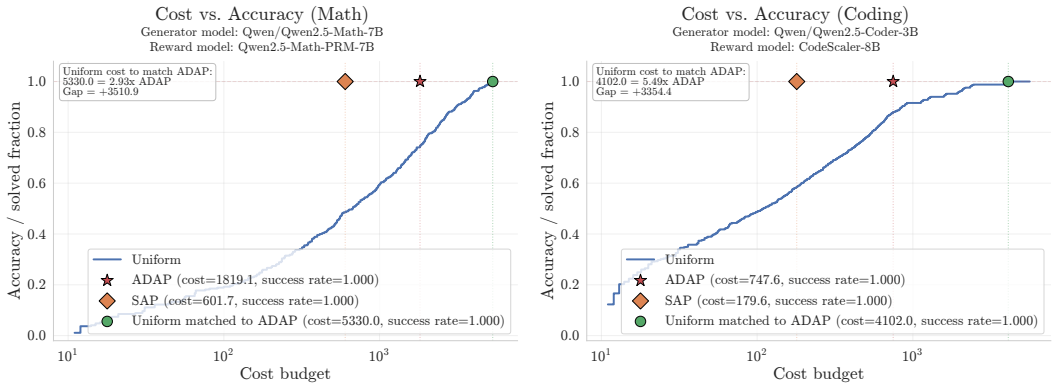
5.3 Results

Table 1 reports mean generation count $\overline{N_{\text{rew}}}$, verification count $\overline{N_{\text{ver}}}$, mean cost, success rate, and cost ratio relative to ADAP for all policies on both tasks.

Table 1: Summary of all policies on Math (HMMT, 27 problems, 10 permutations) and Coding (LiveCodeBench, 83 problems, 10 permutations). Cost ratio is relative to ADAP. DAP$_k$ rows use the stronger variant; results for the weaker variant appear in Appendix B.4. The reported $\overline{N_{\text{rew}}}$ and $\overline{N_{\text{ver}}}$ are mean generation and verification counts over all problem–permutation trials.

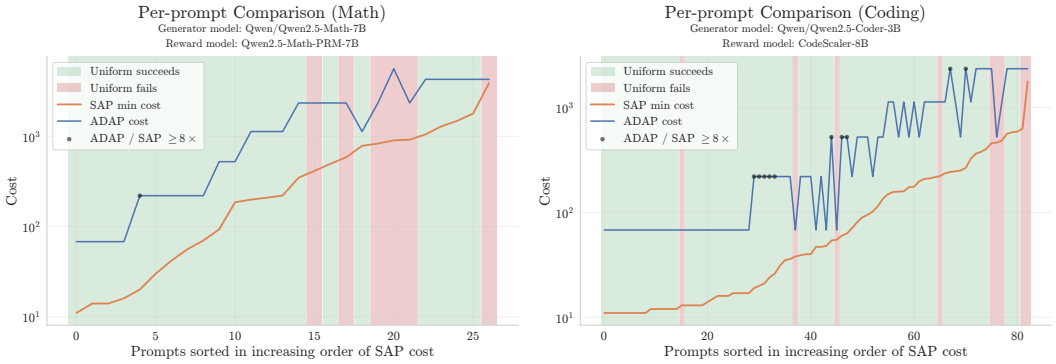
	Math (27 solvable prompts)					Coding (83 solvable prompts)				
Policy	$\overline{N_{\text{rew}}}$	$\overline{N_{\text{ver}}}$	Cost	Ratio	Succ.	$\overline{N_{\text{rew}}}$	$\overline{N_{\text{ver}}}$	Cost	Ratio	Succ.
SAMPLEAWARE	140.7	46.1	601.7	×0.33	1.000	92.4	8.7	179.6	×0.24	1.000
ADAP	273.5	154.6	1819.1	×1.00	1.000	140.4	60.7	747.6	×1.00	1.000
UNIFORM$_{C_{\text{ADAP}}}$	266.0	155.0	1816.0	×1.00	0.744	426.0	32.0	746.0	×1.00	0.880
DAP$_1$	438.8	415.8	4596.8	×2.53	1.000	472.8	331.5	3787.8	×5.07	1.000
DAP$_2$	317.5	248.5	2802.4	×1.54	1.000	305.1	138.7	1692.6	×2.26	1.000
DAP$_3$	268.0	190.7	2175.5	×1.20	1.000	260.0	92.4	1183.9	×1.58	1.000
DAP$_5$	257.9	132.6	1583.6	×0.87	1.000	208.9	59.0	799.0	×1.07	1.000
DAP$_{10}$	208.3	72.5	933.5	×0.51	1.000	170.5	30.3	473.3	×0.63	1.000

Math (HMMT). ADAP achieves 100% success at mean cost 1819.1, while SAMPLEAWARE's mean cost is 601.7 (0.33×). As shown in Figure 3, ADAP's per-trial cost tracks SAMPLEAWARE's closely across the difficulty spectrum. At the same budget, UNIFORM$_{C_{\text{ADAP}}}$ succeeds on only 74.4% of trials, a notable gap that confirms a uniform fixed strategy cannot replicate ADAP's reliability at



(a) Math: Cost vs. Accuracy.

(b) Coding: Cost vs. Accuracy.



(c) Math: Per-prompt comparison across three policies.

(d) Coding: Per-prompt comparison across three policies.

Figure 3: Top: success rate vs. cost budget for the best Uniform strategy (blue step curve), with SAMPLEAWARE (orange diamond) and ADAP (red star) marked at their mean costs. **Bottom:** per-prompt costs of ADAP (blue) and SAMPLEAWARE (orange), sorted by SAMPLEAWARE cost; background shading indicates whether the cost-matched Uniform strategy succeeds (green) or fails (red) on each trial.

equal cost. Among difficulty-stratified baselines, DAP_1 costs $2.53\times$ more than ADAP, and the gap narrows as k grows: DAP_5 costs $0.87\times$ as much as ADAP, while DAP_{10} reaches $0.51\times$. With only 27 problems, ten difficulty classes place on average just two problems per class, making DAP_{10} nearly a per-problem oracle and thus approaching SAMPLEAWARE.

Coding (LiveCodeBench). ADAP achieves 100% success at mean cost 747.6, with SAMPLEAWARE costing 179.6 ($0.24\times$). Figure 3 again shows ADAP's per-trial cost following SAMPLEAWARE's closely. $\text{UNIFORM}_{C_{\text{ADAP}}}$ falls short at 88.0% success, a clear failure mode of uniform strategies on a heterogeneous task. DAP_k costs $5.07\times$ as much as ADAP at $k = 1$, $1.58\times$ at $k = 3$, and $0.63\times$ at $k = 10$; its cost decreases to SAMPLEAWARE at $k = 83$.

6 Learning-Theoretic View of Active Search

In Section 4, monotonicity of the score-conditioned success function gave ADAP a constant-factor guarantee relative to the distribution-aware optimum. We now show that some such structure is necessary: for unstructured $\mathcal{H}$, no sound active-search policy can achieve a uniform constant-factor guarantee. The obstruction is that a positive response may be hidden among many locations that are indistinguishable without verification, while a distribution-aware oracle can focus on the right one immediately. We formalize this using the *centered star number*, a variant of the classical star number of [BHV10; HY15] adapted to finding a single positive witness. We first prove the lower bound in the binary setting, and then extend it to probabilistic concept classes in Section E.1. We also show the seperation and connections between active search and active learning [BBL06; BU16] in Section E.2.

Definition 6.1. Let $\mathcal{Z}$ be an arbitrary input space. Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{Z}}$ be a binary concept class over $\mathcal{Z}$ such that it includes the all zero function which is denoted by h_0. Then, define

$$\mathfrak{s}_0(\mathcal{H}) \;=\; \max\Big\{\, m \in \mathbb{N} \;:\; \exists\, z_1, \ldots, z_m \in \mathcal{Z},\; h_1, \ldots, h_m \in \mathcal{H} \text{ with } h_i(z_j) = \mathbb{1}\,[i = j] \,\Big\}.$$

Algorithm 2 $\mathcal{A}_{\mathrm{CS}}$: Centered-Star Active Search

Input: concept class $\mathcal{H} \in \{0,1\}^{\mathcal{Z}}$, costs $c_{\mathrm{rew}}, c_{\mathrm{ver}} > 0$
Output: a generated point with observed verifier label 1

if $\mathfrak{s}_0 > \frac{c_{\mathrm{ver}}}{c_{\mathrm{rew}}}$ **then**
  while true **do**
    Generate $Z \sim \mathcal{D}$ and pay c_{rew} and Verify Z and pay c_{ver}
    if the verifier returns 1 **then**
      return Z
else
  $n \leftarrow \lceil \frac{c_{\mathrm{ver}}}{c_{\mathrm{rew}}} \mathfrak{s}_0 \rceil$
  while true **do**
    Generate $(Z_1, \ldots, Z_n) \sim \mathcal{D}^{\otimes n}$ and pay nc_{rew}
    Let $S = (Z_1, \ldots, Z_n)$
    For every $h \in \mathcal{H}$, construct $A_h(S) := \{i \in [n] : h(Z_i) = 1\}$
    Choose a *minimum-cardinality hitting set* $I \subseteq [n]$ for $\{A_h(S) : h \in \mathcal{H},\ A_h(S) \neq \emptyset\}$
    for $i \in I$ **do** ▷ see Lemma E.8
      Verify Z_i and pay c_{ver}
      if the verifier returns 1 **then**
        return Z_i

with $\mathfrak{s}_0(\mathcal{H}) = \infty$ if no finite maximum exists.

Equipped with this definition, we state an impossibility result.

Theorem 6.2. *Let $\mathcal{Z}$ be an arbitrary input space. Fix costs $c_{rew}, c_{ver} > 0$. Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{Z}}$ be a class containing the all-zero function, and let $\mathfrak{s}_0(\mathcal{H})$ be its centered star number. For every sound active search policy $\mathcal{A}$ from Definition 2.2, there exists $h^\star \in \mathcal{H}$ and $\mathcal{D} \in \Delta(\mathcal{Z})$ such that*

$$\mathbb{E}[J(\mathcal{A}; h^\star, \mathcal{D})] \geq \frac{1}{4} \min\left\{\mathfrak{s}_0(\mathcal{H}), \frac{c_{ver}}{c_{rew}}\right\} J^\star(h^\star, \mathcal{D}),$$

where $J^\star(h^\star, \mathcal{D})$ is the distribution-aware optimum cost.

Next we discuss the implication of this result: If $\mathfrak{s}_0 = \infty$, then no sound active search policy can have a uniform constant-factor guarantee against the distribution-aware oracle. In particular, in the regime $c_{\mathrm{ver}} \gg c_{\mathrm{rew}}$, the worst-case ratio can be as large as $\Omega(c_{\mathrm{ver}}/c_{\mathrm{rew}})$. This result shows that without assuming structure on h, which corresponds to the case that $\mathfrak{s}_0 = \infty$, it is impossible to develop a policy that satisfies a guarantee analogous to Theorem 4.2.

6.1 Matching Upper Bound

In this part, we complement our lower bound in Theorem 6.2 with an algorithm, $\mathcal{A}_{\mathrm{CS}}$ in Algorithm 2, that achieves a competitive ratio $O\left(\min\left\{\mathfrak{s}_0(\mathcal{H}), \frac{c_{\mathrm{ver}}}{c_{\mathrm{rew}}}\right\}\right)$ compared to the distribution-aware benchmark. Note we do not claim $\mathcal{A}_{\mathrm{CS}}$ is computationally efficient for arbitrary $\mathcal{H}$.

Theorem 6.3. *Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{Z}}$ contain the all zero function. Assume $c_{ver} \geq c_{rew}$. Then for every feasible binary instance $(h^\star, \mathcal{D})$ with $h^\star \in \mathcal{H}$, $\mathcal{A}_{\mathrm{CS}}$ in Algorithm 2 is sound and satisfies*

$$\mathbb{E}[J(\mathcal{A}_{\mathrm{CS}}; h^\star, \mathcal{D})] \leq 6 \min\left\{\mathfrak{s}_0(\mathcal{H}), \frac{c_{ver}}{c_{rew}}\right\} J^\star(h^\star, \mathcal{D}).$$

We first explain the main observation that leads to the matching upper bound. The idea is to use the centered star number to certify an entire generated batch with only a few verifier calls. Given a finite batch $S \in \mathcal{Z}^\star$, each $h \in \mathcal{H}$ induces a positive trace $A_h(S) := \{z \in S : h(z) = 1\}$. A hitting set $Q \subseteq S$ for the nonempty traces has the property that, if the true target has any positive point in S, then it has one in Q. Thus verifying Q either finds a positive or certifies that the whole batch can be discarded. The centered star number bounds the size of such certificates, as shown by Lemma E.8.

Remark 6.4. The characterization of the active search in the binary case in Theorems 6.2 and 6.3 is based on a centered version of the usual star number. The usual star number, that governs distribution-free *active learning* [Han14; HY15], allows the center concept to be arbitrary. Our $\mathfrak{s}_0$ fixes the center

to be the all-zero concept, so $\mathfrak{s}_0(\mathcal{H}) \leq \mathfrak{s}(\mathcal{H})$. This is the right obstruction for active search since the learner does not need to identify the whole target concept, only to find one verified positive. Thus $\mathfrak{s}_0$ captures how many different locations can hide the unique positive against an all-negative baseline. Active learning can be harder because it must distinguish all nearby alternatives around arbitrary centers. In Section E.2, we formally show per-instance active searching is easier than active learning. ◁

7 Conclusion and Limitations

We formulated generate–rank–verify inference as a cost-sensitive active search problem with cheap reward scores and costly final verification, and proposed ADAP, an online policy that adapts generation and verification effort per prompt. Under monotone score-conditioned success probabilities, ADAP achieves a constant-factor guarantee relative to the distribution-aware benchmark; our lower bound shows that some such structure is necessary. The work rests on several simplifying assumptions: the guarantee requires monotonicity, the evaluation covers only feasible prompts, costs are treated as fixed scalars rather than varying with response length, and decisions are based on reward scores alone while in practice verification can exploit response content directly (e.g., detecting syntactic invalidity without invoking a costly checker). Extensions to variable costs, multiple reward models, parallel generation/verification, and richer inductive biases as well as determining the optimal universal approximation constant under monotonicity are natural future directions.

Acknowledgments and Disclosure of Funding

The authors thank Vincent Cohen-Addad for inspiring discussions in the early stages of this project, and Avrim Blum for pointing out connections to the active learning literature. The authors acknowledge the Center for Advanced Research Computing (CARC) at the University of Southern California for providing computing resources that have contributed to the research results reported within this publication. URL: https://carc.usc.edu.

Shaddin Dughmi and Yusuf Hakan Kalayci were supported by the Air Force Office of Scientific Research under award number FA9550-24-1-0261. This work was done while Shaddin Dughmi was on sabbatical as the Carter and Tania Neild visiting professor at Northwestern University, as well as a visiting professor in the Data Science Institute at the University of Chicago. Mahdi Haghifam's work was conducted while he was visiting the Simons Institute for the Theory of Computing. Yusuf Hakan Kalayci's work was conducted while he was a PhD student at the University of Southern California.

References

[BBL06] M.-F. Balcan, A. Beygelzimer, and J. Langford. "Agnostic active learning". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 65–72.

[Bal+26] M.-F. Balcan, A. Blum, K. Fragkia, Z. Li, and D. Sharma. "Online Learnability of Chain-of-Thought Verifiers: Soundness and Completeness Trade-offs". *arXiv preprint arXiv:2603.03538* (2026).

[Bal+25] M.-F. Balcan, A. Blum, Z. Li, and D. Sharma. "On Learning Verifiers and Implications to Chain-of-Thought Reasoning". *Advances in Neural Information Processing Systems* 38 (2025), pp. 154510–154537.

[BHV10] M.-F. Balcan, S. Hanneke, and J. W. Vaughan. "The true sample complexity of active learning". *Machine learning* 80.2 (2010), pp. 111–139.

[BU16] M.-F. Balcan and R. Urner. *Active Learning-Modern Learning Theory.* 2016.

[Ces+97] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Helmbold, R. E. Schapire, and M. K. Warmuth. "How to use expert advice". *Journal of the ACM (JACM)* 44.3 (1997), pp. 427–485.

[Che+23] B. Chen, F. Zhang, A. Nguyen, D. Zan, Z. Lin, J.-G. Lou, and W. Chen. "CodeT: Code Generation with Generated Tests". In: *International Conference on Learning Representations*. 2023.

[Cob+21] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. *Training Verifiers to Solve Math Word Problems*. 2021. arXiv: 2110.14168 [cs.LG].

[Dam+25] M. Damani, I. Shenfeld, A. Peng, A. Bobu, and J. Andreas. "Learning How Hard to Think: Input-Adaptive Allocation of LM Computation". In: *International Conference on Learning Representations*. 2025.

[Dee25] DeepSeek-AI. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025. arXiv: 2501.12948 [cs.CL].

[Gar+12] R. Garnett, Y. Krishnamurthy, X. Xiong, J. G. Schneider, and R. P. Mann. "Bayesian Optimal Active Search and Surveying". In: *Proceedings of the 29th International Conference on Machine Learning*. 2012, pp. 843–850.

[Goo25] Google DeepMind. *Gemini achieves gold-medal level at the International Collegiate Programming Contest World Finals*. Blog post. 2025.

[Han14] S. Hanneke. "Theory of Disagreement-Based Active Learning". *Foundations and Trends in Machine Learning* 7.2–3 (2014), pp. 131–309.

[HY15] S. Hanneke and L. Yang. "Minimax analysis of active learning". *The Journal of Machine Learning Research* 16.1 (2015), pp. 3487–3602.

[HMM25] HMMT. *HMMT February 2024 and 2025 Math Tournaments*. https://www.hmmt.org/www/archive/problems. Harvard-MIT Mathematics Tournament. 2025.

[Hua+25] A. Huang, A. Block, Q. Liu, N. Jiang, A. Krishnamurthy, and D. J. Foster. "Is best-of-n the best of them? coverage, scaling, and optimality in inference-time alignment". *arXiv preprint arXiv:2503.21878* (2025).

[HMZ26] J. Huang, W. Ma, and Z. Zhou. *Optimal Bayesian Stopping for Efficient Inference of Consistent LLM Answers*. 2026. arXiv: 2602.05395 [cs.LG].

[Hui+24] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, K. Dang, Y. Fan, Y. Zhang, A. Yang, R. Men, F. Huang, B. Zheng, Y. Miao, S. Quan, Y. Feng, X. Ren, X. Ren, J. Zhou, and J. Lin. *Qwen2.5-Coder Technical Report*. 2024. arXiv: 2409.12186 [cs.CL].

[Jai+24] N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica. "LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code". *arXiv preprint arXiv:2403.07974* (2024).

[Jia+18] S. Jiang, G. Malkomes, M. Abbott, B. Moseley, and R. Garnett. "Efficient Nonmyopic Batch Active Search". In: *Advances in Neural Information Processing Systems*. Vol. 31. 2018, pp. 1099–1109.

[Jia+17] S. Jiang, G. Malkomes, G. Converse, A. Shofner, B. Moseley, and R. Garnett. "Efficient Nonmyopic Active Search". In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 1714–1723.

[JMG19] S. Jiang, B. Moseley, and R. Garnett. "Cost Effective Active Search". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019, pp. 4880–4889.

[KRD25] Y. Kalayci, V. Raman, and S. Dughmi. *Optimal Stopping vs Best-of-N for Inference-Time Optimization*. 2025. arXiv: 2510.01394 [cs.LG].

[KS94] M. J. Kearns and R. E. Schapire. "Efficient distribution-free learning of probabilistic concepts". *Journal of Computer and System Sciences* 48.3 (1994), pp. 464–497.

[Kha+25] M. Khalifa, R. Agarwal, L. Logeswaran, J. Kim, H. Peng, M. Lee, H. Lee, and L. Wang. *Process Reward Models That Think*. 2025. arXiv: 2504.16828 [cs.CL].

[Kiy+26] S. Kiyani, S. Noorani, G. J. Pappas, and H. Hassani. *When to Trust the Cheap Check: Weak and Strong Verification for Reasoning*. 2026. arXiv: 2602.17633 [cs.LG].

[Kwo+23] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. "Efficient Memory Management for Large Language Model Serving with PagedAttention". In: *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*. 2023.

[Li+22] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, T. Hubert, P. Choy, C. de Masson d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals. "Competition-Level Code Generation with AlphaCode". *Science* 378.6624 (2022), pp. 1092–1097.

[Lig+24] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. "Let's Verify Step by Step". In: *International Conference on Learning Representations*. 2024.

[Ma+25] Z. Ma, X. Zhang, J. Zhang, J. Yu, S. Luo, and J. Tang. "Dynamic Scaling of Unit Tests for Code Reward Modeling". In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2025, pp. 6917–6935.

[Ni+23] A. Ni, S. Iyer, D. Radev, V. Stoyanov, W.-t. Yih, S. I. Wang, and X. V. Lin. "LEVER: Learning to Verify Language-to-Code Generation with Execution". In: *Proceedings of the 40th International Conference on Machine Learning*. 2023, pp. 26106–26128.

[Ope24] OpenAI. *Learning to Reason with LLMs*. Blog post. 2024.

[Qu26] S. Qu. *Adaptive Test-Time Compute Allocation via Learned Heuristics over Categorical Structure*. 2026. arXiv: 2602.03975 [cs.AI].

[Qwe24] Qwen Team. *Qwen2.5 Technical Report*. 2024. arXiv: 2412.15115 [cs.CL].

[RAK25] V. Raman, H. Asi, and S. Kale. *AdaBoN: Adaptive Best-of-N Alignment*. 2025. arXiv: 2505.12050 [cs.LG].

[Roh+25] D. Rohatgi, A. Shetty, D. Saless, Y. Li, A. Moitra, A. Risteski, and D. J. Foster. "Taming imperfect process verifiers: A sampling perspective on backtracking". *arXiv preprint arXiv:2510.03149* (2025).

[Set+25] A. Setlur, C. Nagpal, A. Fisch, X. Geng, J. Eisenstein, R. Agarwal, A. Agarwal, J. Berant, and A. Kumar. "Rewarding Progress: Scaling Automated Process Verifiers for LLM Reasoning". In: *International Conference on Learning Representations*. 2025.

[Sne+25] C. Snell, J. Lee, K. Xu, and A. Kumar. "Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning". In: *International Conference on Learning Representations*. 2025.

[Wan+25a] G. Wan, Y. Wu, H. Wang, S. Zhao, J. Chen, and S. Li. "Derailer-Rerailer: Adaptive Verification for Efficient and Reliable Language Model Reasoning". In: *Findings of the Association for Computational Linguistics: ACL 2025*. 2025.

[Wan+25b] G. Wan, Z. S. Xu, S. Zorc, M. Baucells, M. Hu, H. Wang, and S. Li. *BEACON: Bayesian Optimal Stopping for Efficient LLM Sampling*. 2025. arXiv: 2510.15945 [cs.CL].

[Wan+24] P. Wang, L. Li, Z. Shao, R. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. 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)*. 2024, pp. 9426–9439.

[Yan+24] A. Yang, B. Zhang, B. Hui, B. Gao, B. Yu, C. Li, D. Liu, J. Tu, J. Zhou, J. Lin, K. Lu, M. Xue, R. Lin, T. Liu, X. Ren, and Z. Zhang. *Qwen2.5-Math Technical Report: Toward Mathematical Expert Model via Self-Improvement*. 2024. arXiv: 2409.12122 [cs.AI].

[Zha+26] Z. Zhai, B. Li, B. Xiao, M. Li, and X. Wang. *Adaptive Test-Time Compute Allocation for Reasoning LLMs via Constrained Policy Optimization*. 2026. arXiv: 2604.14853 [cs.LG].

[Zha+25a] L. Zhang, A. Hosseini, H. Bansal, M. Kazemi, A. Kumar, and R. Agarwal. "Generative Verifiers: Reward Modeling as Next-Token Prediction". In: *International Conference on Learning Representations*. 2025.

[Zha+25b] Z. Zhang, C. Zheng, Y. Wu, B. Zhang, R. Lin, B. Yu, D. Liu, J. Zhou, and J. Lin. "The Lessons of Developing Process Reward Models in Mathematical Reasoning". In: *Findings of the Association for Computational Linguistics: ACL 2025*. 2025, pp. 10495–10516.

[Zhu+26] X. Zhu, X. Zhou, B. Zhu, H. Hu, M. Du, H. Zhang, H. Wang, and Z. Guo. *CodeScaler: Scaling Code LLM Training and Test-Time Inference via Execution-Free Reward Models*. 2026. arXiv: 2602.17684 [cs.CL].

Contents

1 Introduction **1**

1.1 Contributions 2

2 Problem Setup **3**

2.1 Active Search 3

3 Distribution-Aware Benchmark **4**

4 ADAP: Adaptive Policy **5**

4.1 Main ideas behind ADAP 6

5 Experiments **6**

5.1 Setup 6

5.2 Baselines 7

5.3 Results 7

6 Learning-Theoretic View of Active Search **8**

6.1 Matching Upper Bound 9

7 Conclusion and Limitations **10**

A Related Work **14**

B Additional Experiments **15**

B.1 Computational Resources 15

B.2 Reward Signal Validation 15

B.3 Robust difficulty-class benchmark 17

B.4 Ablations 17

C Appendix of Section 3 **18**

D Appendix of Section 4 **21**

E Appendix of Section 6 **25**

E.1 Extension to Probabilistic Concept Classes 26

E.2 Connection Between Active Learning and Active Search 26

E.3 Proof of Theorem 6.3 28

A Related Work

Generate–rank–verify foundations. Many inference-time reasoning systems follow a generate–evaluate–select pattern. In math reasoning, trained outcome verifiers rank complete solutions [Cob+21], while process reward models and generative verifiers score intermediate or complete reasoning traces [Lig+24; Wan+24; Set+25; Zha+25a; Kha+25; Zha+25b]. In code generation, AlphaCode used large-scale sampling followed by execution-based filtering, clustering, and selection [Li+22]; CodeT and LEVER use generated tests or execution-aware learned verifiers to rank programs [Che+23; Ni+23]; and CodeRM studies dynamic scaling of generated unit tests to improve reward quality [Ma+25]. These works motivate our generate–rank–verify abstraction, but mainly study verifier design, empirical scaling, or engineered selection pipelines.

Adaptive test-time compute allocation. A line of work studies how to spend inference-time compute adaptively. A common theme is that uniform Best-of-N sampling is inefficient because the value of additional samples depends on input difficulty [Sne+25]. Some methods allocate compute across inputs using predicted reward curves [Dam+25], constrained optimization under an average budget [Zha+26], or prompt-level Best-of-N budget allocation [RAK25]. Others adapt computation within a single input, stopping generation when further samples are unlikely to improve reward [KRD25; Wan+25b], or when answer agreement is sufficiently high in self-consistency sampling [HMZ26]. Our focus is different: the policy must decide how to trade off generating additional against verifying available candidates.

Selective and imperfect verification. A related line studies how to use imperfect or costly verification signals. Weak–strong verification policies decide when a cheap noisy verifier is sufficient and when to defer to a stronger verifier [Kiy+26], while Derailer–Rerailer uses a lightweight stability check to trigger more expensive reasoning only when needed [Wan+25a]. Closest to our setting, Qu [Qu26] studies a verification-cost-limited regime and allocates verifier calls across intermediate reasoning states using feasibility gates, learned pre-verification scores, and uncertainty. In contrast, our verifier is treated as the final certification mechanism, and the main question is how costly certification should be interleaved with generation and reward scoring.

Active search. Active search studies discovery-oriented label querying, where the goal is to find positives rather than to learn a globally accurate classifier. Classical Bayesian active search is usually pool-based. A fixed unlabeled pool is given in advance, and a known probabilistic model or Bayesian posterior over labels guides which points to query under a labeling budget [Gar+12; Jia+17; Jia+18]. The closest formulation to ours is cost-effective active search [JMG19], which minimizes labeling cost to find a prescribed number of positives from a fixed pool. Our setting keeps the discovery objective of active search, but the candidate pool is generated online by sampling from a language model and scoring samples with a reward model. Because the score distribution and reward–verifier relationship are unknown, the policy cannot rely on a known posterior over a fixed pool. It must decide both which candidates to verify and when to generate more scored candidates, creating a generation–verification tradeoff absent from standard fixed-pool active search.

Learning-Theoretic View of the Challenges in LLMs. A related line of work studies the learnability of verifiers when we have chain-of-thought traces. [Bal+25] introduce PAC-style models for learning Chain-of-Thought verifiers, including simple verifiers that generalize on traces from a fixed distribution and stronger trustable verifiers that aim to reject faulty reasoning traces more robustly. [Bal+26] develop an online learning framework for Chain-of-Thought verification, emphasizing the asymmetric costs of soundness errors (accepting faulty reasoning) and completeness errors (rejecting correct reasoning), and characterize the optimal mistake tradeoffs using soundness-completeness variants of the Littlestone dimension. Our work addresses a complementary inference-time decision problem. Rather than learning the verifier, we assume access to a final certifying verifier, such as exact answer checking, hidden-test execution, or proof verification, and treat each verifier call as costly. We also assume access to a cheap but noisy reward model that can rank candidate responses. The goal is therefore not to learn a globally reliable verifier, but to adaptively decide how many responses to generate and which high-reward candidates to verify in order to find a verified-correct response at minimum cost.

Inference with Imperfect Proxies. Recent theoretical works analyze inference guided by imperfect reward models. [Hua+25] quantify proxy imperfection via squared error relative to a hypothetical true reward, showing that Best-of-N sampling overexploits tail errors; crucially, their setting relies

entirely on the proxy and lacks a final verifier. [Roh+25] formalize process verifiers as approximate value functions, demonstrating how local evaluation errors compound over long generation horizons. We capture proxy imperfection through a fundamentally different lens. Rather than assuming the proxy is calibrated to a true reward or tracks a process value function, we treat it strictly as a cheap, uncalibrated ranking signal. For each prompt x, the generator and proxy induce an unknown score distribution $\mathcal{D}_x$, while an exact but costly final verifier defines an unknown conditional success probability $h_x^\star(r) = \Pr(V_x = 1 \mid R_x = r)$. This decouples the noisy proxy from the ground-truth verifier, shifting the theoretical focus from bounding proxy approximation errors to cost-sensitive active search.

B Additional Experiments

B.1 Computational Resources

All experiments run on a single node with two NVIDIA L40S GPUs (48 GB each). Generation and math reward scoring use vLLM [Kwo+23] with TP=2 (Tensor Parallelism); coding reward scoring uses CodeScaler-8B via PyTorch on a single GPU; coding verification is CPU-only with 64 parallel workers and a 15-second per-test timeout. All figures below are estimates; exact per-stage timing was not logged.

Stage	Details	Estimated cost
Math generation	3 models $\times$ 60 problems $\times$ 512 samples; 8–10 h/model on 2$\times$L40S	$\sim$48–60 GPU-hour
Math reward scoring	Qwen2.5-Math-PRM-7B via vLLM, 92,160 samples	$\sim$4–6 GPU-hour
Coding generation	Qwen2.5-Coder-3B, TP=2, 329 $\times$ 512 = 168,448 samples, $\sim$750 tok/sample	$\sim$26–40 GPU-hour
Coding reward scoring	CodeScaler-8B, single GPU, 42,496 samples, batch size 32	$\sim$2–3 GPU-hour
Total GPU		**$\sim$85–110 GPU-hour**
Coding verification	42,496 programs, 64 workers, $\sim$20–30 CPU-s/sample (wall time $\approx$4–6 h)	$\sim$250–400 CPU-hour

B.2 Reward Signal Validation

The adaptive policy assumes that the reward model is more likely to assign higher scores to correct samples than to incorrect ones. We verify this directly for coding; math diagnostics for alternative generators appear in Appendix B.4.1.

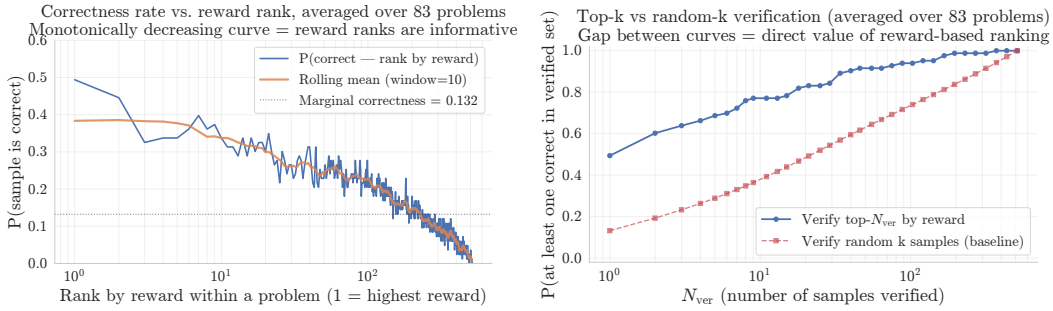
Rank vs. correctness. For each problem we sort samples in decreasing reward order and record correctness at each rank position. Figure 4a plots the empirical correctness probability averaged across the 83 coding problems and Figure 5a plots the same result averaged across the 27 math problems. The curve is monotonically decreasing: the top-ranked sample is several times more likely to be correct than a uniformly random sample. Our policy requires only this rank monotonicity, not calibrated reward probabilities.

Top-k coverage. Figure 4b plots, for each k, the probability that the top-k-by-reward set contains at least one correct sample, alongside the random-k baseline (hypergeometric distribution). Using the reward model substantially raises coverage even at $k = 1$, and the gap remains positive throughout the small-k regime where the adaptive policy operates.

Per-problem AUC distribution. To assess whether the reward signal is consistent across problems, we compute a per-problem AUC:

$$\mathrm{AUC}_p = \Pr\big(R > R' \;\big|\; V = 1,\, V' = 0\big) + \tfrac{1}{2}\Pr\big(R = R' \;\big|\; V = 1,\, V' = 0\big),$$

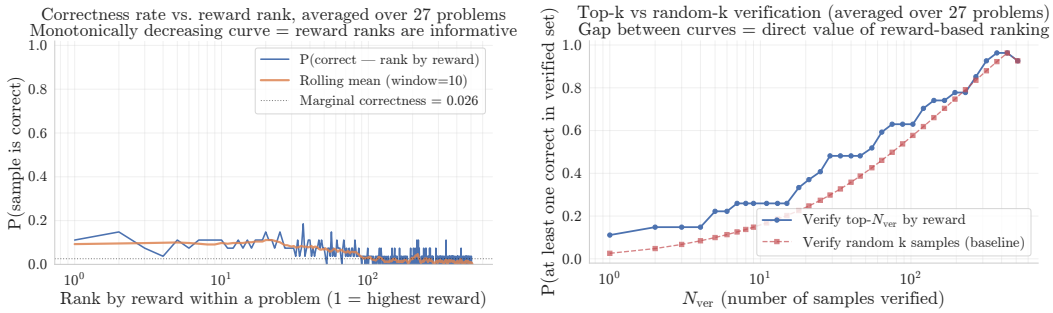
where (R, V) and (R', V') are independent draws from the joint distribution of prompt p (i.e. $R = \mathsf{RM}(p, Y)$, $V = \mathsf{Ver}(p, Y)$ with $Y \sim \pi(\cdot \mid p)$, and likewise for the primed copy).



(a) Correctness probability at each reward rank.

(b) Top-k vs. random-k coverage.

Figure 4: Reward signal validation on coding. Left: empirical correctness probability at each reward rank, averaged over 83 problems (orange: rolling mean). Right: probability that the top-k by reward contains a correct sample (blue) vs. k uniform random samples (red dashed).



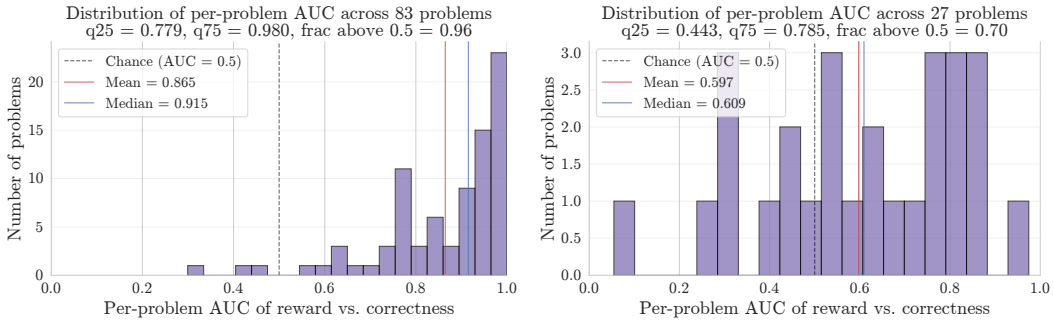
(a) Correctness probability at each reward rank.

(b) Top-k vs. random-k coverage.

Figure 5: Reward signal validation on math. Left: empirical correctness probability at each reward rank, averaged over 27 problems (orange: rolling mean). Right: probability that the top-k by reward contains a correct sample (blue) vs. k uniform random samples (red dashed).

For **coding**, the distribution is strongly right-skewed (mean: 0.865, median: 0.915, IQR: $[0.779, 0.980]$, 96% above chance), confirming that CodeScaler-8B reliably ranks correct samples higher on the vast majority of problems.

For **math**, the signal is weaker (mean: 0.597, median: 0.609, IQR: $[0.443, 0.785]$, 70% above chance), yet the mean AUC remains meaningfully above chance and the top-k coverage curve (Figure 5) still lies above the random baseline throughout, confirming that even a modest reward signal suffices to guide verification.



(a) Coding (83 problems).

(b) Math (27 problems).

Figure 6: Per-problem AUC of reward vs. correctness. Each bar counts the number of problems whose reward model achieves that AUC when ranking samples by score. The dashed vertical line marks chance (AUC = 0.5); red and blue verticals mark the mean and median. Coding rewards are strongly discriminative (mean 0.865, 96% above chance); math rewards are weaker and more variable (mean 0.597, 70% above chance).

B.3 Robust difficulty-class benchmark

The complementary benchmark $\mathrm{DAP}_k^{\mathrm{robust}}$ jointly chooses one contiguous partition $G_1, \ldots, G_k$ of the same fixed empirical-difficulty order and one pair $(m_i, v_i) = ((N_{\mathrm{rew}})_i, (N_{\mathrm{ver}})_i)$ per class, with integers $1 \le v_i \le m_i \le 512$. It minimizes

$$\frac{1}{P}\sum_{i=1}^{k} |G_i|(c_{\mathrm{rew}} m_i + c_{\mathrm{ver}} v_i), \qquad S_{p,t}(m_i, v_i) = 1 \quad \text{for all } i \in [k],\ p \in G_i,\ t \in [T].$$

Here P is the number of problems, $T = 10$ is the number of sampled orderings, and $S_{p,t}(m, v)$ indicates whether verifying the v highest-reward responses among the first m draws succeeds on problem p in ordering t. This is the original all-orderings DAP constraint. Relative to the per-ordering oracle, its choices have less freedom, so its optimal cost is at least as large. Both benchmarks use hidden verifier outcomes and are hindsight comparators. Its cost depends on T. Robust DAP protects only against the T observed orderings, with no guarantee on unseen ones. At $k = P$, it still needs one budget per prompt that succeeds on every ordering; it need not equal SAMPLEAWARE. Counts, as in Table 1, are weighted by class size and averaged over all prompt–ordering trials.

B.4 Ablations

B.4.1 Alternative math generators

The main body reports math results with `Qwen2.5-Math-7B` as the generator. We additionally evaluate `Qwen2.5-14B` (a stronger, non-math-specialized base model) and `DeepSeek-R1-Distill-Qwen-7B` model. Both use identical sampling settings ($N = 512$, temperature 0.7, top-p 0.95), the same reward model (`Qwen2.5-Math-PRM-7B`), and exact answer-string match as the verifier.

Table 2 reports the number of solvable problems and mean cost of each policy per generator. The qualitative picture is unchanged across generators: rank-order monotonicity of the reward signal holds in all cases, and ADAP achieves 100% success at substantially lower cost than DAP_1, demonstrating that the approach is not specific to the primary generator and generalizes reliably across models of varying size and specialization.

Table 2: $c_{\mathrm{ver}}/c_{\mathrm{rew}} = 10$. Per-generator results on HMMT. Mean cost averaged over solvable problems and 10 permutations each. The ratio column shows cost relative to ADAP for the same generator.

	Qwen2.5-Math-7B		Qwen2.5-14B		DeepSeek-R1-7B	
Policy	Cost	Ratio	Cost	Ratio	Cost	Ratio
# Solvable	27		22		22	
SAMPLEAWARE	601.7	×0.33	487.5	×0.32	476.6	×0.33
ADAP	1819.1	×1.00	1508.5	×1.00	1443.6	×1.00
DAP_1	4596.8	×2.53	4334.2	×2.87	4058.8	×2.81
DAP_2	2802.4	×1.54	2029.6	×1.35	1556.9	×1.08
DAP_3	2175.5	×1.20	1430.2	×0.95	1150.5	×0.80
DAP_5	1583.6	×0.87	1015.8	×0.67	840.4	×0.58
DAP_{10}	933.5	×0.51	641.6	×0.43	566.3	×0.39
$\mathrm{DAP}_1^{\mathrm{robust}}$	5330.0	×2.93	4661.0	×3.09	5532.0	×3.83
$\mathrm{DAP}_2^{\mathrm{robust}}$	3730.0	×2.05	2192.5	×1.45	1980.3	×1.37
$\mathrm{DAP}_3^{\mathrm{robust}}$	3014.6	×1.66	1716.7	×1.14	1691.6	×1.17
$\mathrm{DAP}_5^{\mathrm{robust}}$	2403.1	×1.32	1350.8	×0.90	1393.6	×0.97
$\mathrm{DAP}_{10}^{\mathrm{robust}}$	1690.7	×0.93	1134.0	×0.75	1149.6	×0.80
$\mathrm{UNIFORM}_{C_{\mathrm{ADAP}}}$ success	0.744		0.864		0.827	

B.4.2 Cost-ratio robustness

The main body fixes $c_{\mathrm{ver}}/c_{\mathrm{rew}} = 10$. We repeat the comparison for $c_{\mathrm{ver}}/c_{\mathrm{rew}} \in \{1, 10, 20, 30\}$ on both tasks to verify that ADAP's advantage is not specific to this calibration. Tables 3 and 4 summarize the results.

ADAP consistently achieves lower cost than DAP_1 at every ratio on both tasks, and the advantage on coding grows as verification becomes more expensive: DAP_1 costs $3.3\times$ as much as ADAP at

$c_{\text{ver}}/c_{\text{rew}} = 1$, rising to $6.7\times$ at ratio 30. The UNIFORM$_{C_{\text{ADAP}}}$ rows confirm that a fixed policy given the same total budget as ADAP achieves only 73.7–90.4% success, underscoring that the savings come from genuine adaptivity rather than simply spending less. Notably, ADAP has lower cost than DAP$_{10}^{\text{robust}}$ on coding at all ratios, without requiring any offline difficulty estimation or problem partitioning. The stronger DAP$_{10}$ is cheaper than ADAP on both tasks at all four ratios.

Table 3: Cost-ratio ablation on math (HMMT, 27 problems, 10 permutations each). The ratio column shows cost relative to ADAP at the same configuration.

	$c_{\text{ver}} = 1$		$c_{\text{ver}} = 10$		$c_{\text{ver}} = 20$		$c_{\text{ver}} = 30$	
Policy	Cost	Ratio	Cost	Ratio	Cost	Ratio	Cost	Ratio
SAMPLEAWARE	176.6	$\times$0.51	601.7	$\times$0.33	1058.6	$\times$0.33	1511.8	$\times$0.32
ADAP	349.5	$\times$1.00	1819.1	$\times$1.00	3203.1	$\times$1.00	4794.4	$\times$1.00
UNIFORM$_{C_{\text{ADAP}}}$ success	0.737		0.744		0.741		0.741	
DAP$_1$	854.6	$\times$2.45	4596.8	$\times$2.53	8754.8	$\times$2.73	12912.8	$\times$2.69
DAP$_2$	553.0	$\times$1.58	2802.4	$\times$1.54	5272.3	$\times$1.65	7738.4	$\times$1.61
DAP$_3$	449.4	$\times$1.29	2175.5	$\times$1.20	4073.3	$\times$1.27	5967.1	$\times$1.24
DAP$_5$	359.0	$\times$1.03	1583.6	$\times$0.87	2902.8	$\times$0.91	4215.0	$\times$0.88
DAP$_{10}$	251.3	$\times$0.72	933.5	$\times$0.51	1649.2	$\times$0.51	2356.2	$\times$0.49
DAP$_1^{\text{robust}}$	992.0	$\times$2.84	5330.0	$\times$2.93	10150.0	$\times$3.17	14970.0	$\times$3.12
DAP$_2^{\text{robust}}$	740.9	$\times$2.12	3730.0	$\times$2.05	6950.0	$\times$2.17	10170.0	$\times$2.12
DAP$_3^{\text{robust}}$	651.9	$\times$1.87	3014.6	$\times$1.66	5521.6	$\times$1.72	8028.7	$\times$1.67
DAP$_5^{\text{robust}}$	567.6	$\times$1.62	2403.1	$\times$1.32	4299.8	$\times$1.34	6196.4	$\times$1.29
DAP$_{10}^{\text{robust}}$	476.6	$\times$1.36	1690.7	$\times$0.93	2913.9	$\times$0.91	4129.9	$\times$0.86

Table 4: Cost-ratio ablation on coding (LiveCodeBench, 83 problems, 10 permutations each). The ratio column shows cost relative to ADAP at the same configuration.

	$c_{\text{ver}} = 1$		$c_{\text{ver}} = 10$		$c_{\text{ver}} = 20$		$c_{\text{ver}} = 30$	
Policy	Cost	Ratio	Cost	Ratio	Cost	Ratio	Cost	Ratio
SAMPLEAWARE	97.6	$\times$0.40	179.6	$\times$0.24	265.2	$\times$0.25	348.8	$\times$0.22
ADAP	241.4	$\times$1.00	747.6	$\times$1.00	1074.4	$\times$1.00	1564.1	$\times$1.00
UNIFORM$_{C_{\text{ADAP}}}$ success	0.795		0.880		0.880		0.904	
DAP$_1$	804.3	$\times$3.33	3787.8	$\times$5.07	7102.8	$\times$6.61	10417.8	$\times$6.66
DAP$_2$	409.1	$\times$1.69	1692.6	$\times$2.26	3072.6	$\times$2.86	4452.2	$\times$2.85
DAP$_3$	319.4	$\times$1.32	1183.9	$\times$1.58	2100.2	$\times$1.95	3012.3	$\times$1.93
DAP$_5$	248.8	$\times$1.03	799.0	$\times$1.07	1378.1	$\times$1.28	1947.0	$\times$1.24
DAP$_{10}$	186.1	$\times$0.77	473.3	$\times$0.63	769.0	$\times$0.72	1059.5	$\times$0.68
DAP$_1^{\text{robust}}$	871.0	$\times$3.61	4102.0	$\times$5.49	7692.0	$\times$7.16	11282.0	$\times$7.21
DAP$_2^{\text{robust}}$	498.3	$\times$2.06	2162.1	$\times$2.89	3889.7	$\times$3.62	5617.3	$\times$3.59
DAP$_3^{\text{robust}}$	413.4	$\times$1.71	1501.5	$\times$2.01	2604.7	$\times$2.42	3708.0	$\times$2.37
DAP$_5^{\text{robust}}$	335.3	$\times$1.39	1183.3	$\times$1.58	1984.7	$\times$1.85	2784.8	$\times$1.78
DAP$_{10}^{\text{robust}}$	281.9	$\times$1.17	764.8	$\times$1.02	1222.2	$\times$1.14	1661.4	$\times$1.06

C Appendix of Section 3

Lemma C.1. *Fix an active search instance $(h^\star, \mathcal{D})$ from Definition 2.1 with $R \sim \mathcal{D}$. Consider the class $\Pi_{\text{str}}(\mathcal{D}, h^\star)$ of streaming policies defined as follows. At each round i, the policy draws a fresh score $R_i \sim \mathcal{D}$, pays c_{rew}, observes R_i, and then chooses an action $A_i \in \{\textsf{discard}, \textsf{verify}\}$ as a possibly randomized function of the past history and the current score R_i. If $A_i = \textsf{discard}$, the candidate is discarded permanently. If $A_i = \textsf{verify}$, the policy pays c_{ver} and observes $V_i \mid R_i \sim \text{Bernoulli}(h^\star(R_i))$. The policy stops only when it verifies a candidate with $V_i = 1$. Then there exists a unique $\tau^\star \in (0,1)$ satisfying*

$$c_{ver}\,\mathbb{E}_{R\sim D}[(h^\star(R) - \tau^\star)_+] = \tau^\star c_{rew}.$$

Moreover,

$$J^\star_{\text{str}}(h, D) = \frac{c_{ver}}{\tau^\star}.$$

An optimal streaming policy is the threshold rule

$$A^\star(r) = \begin{cases} \textsf{verify}, & h(r) \geq \tau^\star, \\ \textsf{discard}, & h(r) < \tau^\star. \end{cases}$$

Proof. Let $(R_i, V_i)_{i\geq 1}$ be i.i.d. copies of (R, V). A (possibly randomized, history-dependent) policy π chooses at each round i an action $A_i \in \{\textsf{discard}, \textsf{verify}\}$ after observing R_i (and the past history). Let

$$T := \inf\{i \geq 1 : \ A_i = \textsf{verify} \text{ and } V_i = 1\}$$

be the stopping round, and define the total cost

$$\mathrm{Cost}(\pi) := \sum_{i=1}^{T} \Big(c_{\text{rew}} + c_{\text{ver}} \mathbf{1}\{A_i = \textsf{verify}\} \Big), \qquad J(\pi) := \mathbb{E}[\mathrm{Cost}(\pi)], \qquad J^* := \inf_{\pi} J(\pi).$$

Fix a round, and condition on having already paid c_{rew} and observed $R = r$.

- If we $\textsf{discard}$, we move to the start of the next round, whose optimal expected remaining cost is J^*.
- If we $\textsf{verify}$, we pay c_{ver} and observe V. With probability $h(r)$ we stop immediately; with probability $1 - h(r)$ we fail and return to a fresh round with optimal expected remaining cost J^*. Thus the continuation cost is

$$c_{\text{ver}} + (1 - h(r))J^*.$$

Therefore, verifying is optimal iff

$$c_{\text{ver}} + (1 - h(r))J^* \leq J^* \quad \Longleftrightarrow \quad h(r) \geq \frac{c_{\text{ver}}}{J^*}.$$

Define

$$\tau^* := \frac{c_{\text{ver}}}{J^*} \in (0, 1]. \tag{1}$$

This shows there exists an optimal policy that verifies exactly when $h(r) \geq \tau^*$.

To characterize τ^*, we use Bellman equation. At the start of a fresh round, we pay c_{rew}, observe R, and then choose the smaller of discarding (continuation cost J^*) and verifying (continuation cost $c_{\text{ver}} + (1 - h(R))J^*$). Thus J^* satisfies the Bellman identity

$$J^* = c_{\text{rew}} + \mathbb{E}\Big[\min\{J^*, \ c_{\text{ver}} + (1 - h(R))J^*\}\Big]. \tag{2}$$

For any $J > 0$ and $x \in [0, 1]$,

$$\min\{J, \ c_{\text{ver}} + (1 - x)J\} = J - (xJ - c_{\text{ver}})_+,$$

where $(u)_+ := \max\{u, 0\}$. Apply this with $x = h(R)$ and $J = J^*$ in (2):

$$J^* = c_{\text{rew}} + \mathbb{E}\big[J^* - (h(R)J^* - c_{\text{ver}})_+\big] \quad \Longleftrightarrow \quad c_{\text{rew}} = \mathbb{E}\big[(h(R)J^* - c_{\text{ver}})_+\big].$$

Using $\tau^* = c_{\text{ver}}/J^*$ from (1), we rewrite

$$(h(R)J^* - c_{\text{ver}})_+ = J^*(h(R) - \tau^*)_+,$$

and hence

$$c_{\text{rew}} = J^* \, \mathbb{E}\big[(h(R) - \tau^*)_+\big] = \frac{c_{\text{ver}}}{\tau^*} \, \mathbb{E}\big[(h(R) - \tau^*)_+\big].$$

Multiplying both sides by τ^* gives

$$\tau^* c_{\text{rew}} = c_{\text{ver}} \, \mathbb{E}\big[(h(R) - \tau^*)_+\big]. \tag{3}$$

Also, (1) yields

$$J^* = \frac{c_{\text{ver}}}{\tau^*}.$$

Then, we show that $\tau^\star$ is unique and exists. Define for $\tau \in [0,1]$,

$$g(\tau) := c_{\text{ver}} \int_{\{r:\, h(r)\geq\tau\}} (h(r)-\tau)\mathrm{d}\mathcal{D}(r) - \tau c_{\text{rew}} = c_{\text{ver}}\mathbb{E}\big[(h(R)-\tau)_+\big] - \tau c_{\text{rew}}.$$

Then g is continuous. Moreover, for $0 \leq \tau_1 < \tau_2 \leq 1$,

$$\begin{aligned} g(\tau_2) - g(\tau_1) &= c_{\text{ver}}\Big(\mathbb{E}[(h(R)-\tau_2)_+] - \mathbb{E}[(h(R)-\tau_1)_+]\Big) - (\tau_2-\tau_1)c_{\text{rew}} \\ &\leq -(\tau_2-\tau_1)c_{\text{rew}} < 0, \end{aligned}$$

so g is strictly decreasing. Also,

$$g(0) = c_{\text{ver}}\mathbb{E}[h(R)] = c_{\text{ver}}\Pr(V=1) > 0, \qquad g(1) = 0 - c_{\text{rew}} < 0.$$

By the intermediate value theorem there exists a root $\tau^* \in (0,1]$, and by strict monotonicity it is unique.

□

Lemma C.2. *Fix an active search instance $(h, \mathcal{D})$ from Definition 2.1. Among all distribution-aware pool-based policies in the sense of Definition 2.2, there exists an optimal policy that never stores candidates for later use. In particular, the streaming threshold policy of Lemma C.1 is optimal within this broader class.*

Proof. Let $J^\star = c_{\text{ver}}/\tau^\star$ be the optimal streaming value from Lemma C.1. Verifying a sequence of candidates is equivalent to verifying them one at a time, so it suffices to compare the two elementary actions: (i) generating a finite batch, and (ii) verifying a single stored candidate.

The candidate continuation value. Given a finite pool P, let $r_{(1)}, \ldots, r_{(k)}$ enumerate its above-threshold elements in decreasing order of h:

$$h(r_{(1)}) \geq \cdots \geq h(r_{(k)}) > \tau^\star,$$

with $k = 0$ if no such elements exist. Define

$$W(P) := c_{\text{ver}} \sum_{i=1}^{k} \prod_{\ell<i} \big(1 - h(r_{(\ell)})\big) \;+\; J^\star \prod_{i=1}^{k} \big(1 - h(r_{(i)})\big). \tag{4}$$

This is the expected cost of the policy that verifies the above-threshold candidates in decreasing order of h, discards the rest, and falls back to a fresh start (value $J^\star$) if all verifications fail. We show that $W(P)$ is the optimal continuation value from P by establishing two structural properties of W and then comparing the elementary actions.

Step 1: $W(P) \leq J^\star$. Set $p_i = h(r_{(i)})$ and define $U_{k+1} = J^\star$, $U_i = c_{\text{ver}} + (1-p_i)U_{i+1}$, so that $W(P) = U_1$. Since $p_i > \tau^\star = c_{\text{ver}}/J^\star$, backward induction on i gives

$$U_i \leq c_{\text{ver}} + (1-p_i)J^\star = J^\star - (p_i J^\star - c_{\text{ver}}) < J^\star,$$

and hence $W(P) \leq J^\star$.

Step 2: Insertion bound. For every deterministic score r,

$$W(P) - W(P \cup \{r\}) \leq \big(h(r)J^\star - c_{\text{ver}}\big)_+. \tag{5}$$

If $h(r) \leq \tau^\star$, then r does not appear in (4), so $W(P \cup \{r\}) = W(P)$ and the bound is immediate. Suppose instead that $p := h(r) > \tau^\star$, and insert r into the ordered above-threshold list. Let $S \leq 1$ denote the probability that all candidates preceding r fail, and let U denote the tail value at the insertion point before adding r. A direct calculation from (4) yields

$$W(P) - W(P \cup \{r\}) = S\big(pU - c_{\text{ver}}\big).$$

The candidates following r all have success probability at most p, and the fall-back value satisfies $J^\star \geq c_{\text{ver}}/p$; a backward induction along the tail then gives $c_{\text{ver}}/p \leq U \leq J^\star$. Combined with $S \leq 1$, this yields $0 \leq S(pU - c_{\text{ver}}) \leq pJ^\star - c_{\text{ver}}$, proving (5).

Step 3: No first action improves on $W(P)$. We bound the expected cost of each elementary action from P.

Generate. Suppose the policy generates a batch $R_1, \dots, R_m \sim \mathcal{D}$. Iterating (5),

$$W(P \cup \{R_1, \dots, R_m\}) \geq W(P) - \sum_{i=1}^{m} \big(h(R_i)J^\star - c_{\text{ver}}\big)_+.$$

The fixed-point identity from Lemma C.1, $\tau^\star c_{\text{rew}} = c_{\text{ver}}\,\mathbb{E}[(h(R) - \tau^\star)_+]$, is equivalent to $c_{\text{rew}} = \mathbb{E}\big[(h(R)J^\star - c_{\text{ver}})_+\big]$ after multiplying by $J^\star$. Therefore, taking expectation,

$$mc_{\text{rew}} + \mathbb{E}\big[W(P \cup \{R_1, \dots, R_m\})\big] \;\geq\; W(P) + mc_{\text{rew}} - m\,\mathbb{E}\big[(h(R)J^\star - c_{\text{ver}})_+\big] \;=\; W(P).$$

Hence generating first cannot improve on $W(P)$.

Verify a below-threshold candidate. If $h(r) \leq \tau^\star$, then $W(P \setminus \{r\}) = W(P)$, and the expected cost of verifying r is

$$c_{\text{ver}} + (1 - h(r))W(P) \;=\; W(P) + c_{\text{ver}} - h(r)W(P) \;\geq\; W(P),$$

where the last inequality uses $W(P) \leq J^\star$ and $h(r) \leq \tau^\star = c_{\text{ver}}/J^\star$, so that $h(r)W(P) \leq c_{\text{ver}}$.

Verify an above-threshold candidate. Among such candidates, verifying in decreasing order of h is optimal. Indeed, for $p \geq q$, verifying p before q costs $c_{\text{ver}} + (1-p)\big(c_{\text{ver}} + (1-q)U\big)$, whereas the reverse order costs $c_{\text{ver}} + (1-q)\big(c_{\text{ver}} + (1-p)U\big)$; the difference is $c_{\text{ver}}(q-p) \leq 0$. The decreasing-$h$ order attains exactly $W(P)$ by construction.

Step 4: Optimality of $W(P)$. Steps 1–3 show that every elementary action has expected continuation cost at least $W(P)$. To extend this one-step comparison to arbitrary policies, fix any pool-based policy π starting from P. Let C_t denote the cost accumulated after t decisions, P_t the pool before the t-th decision, and T the first successful verification time. By induction on t,

$$W(P) \;\leq\; \mathbb{E}_\pi\Big[C_t + W(P_t)\,\mathbf{1}\{T > t\}\Big] \qquad \forall t \geq 1.$$

If $\mathbb{E}_\pi[C_T] = \infty$, there is nothing to prove. Otherwise $T < \infty$ almost surely, since each action costs at least $\min\{c_{\text{rew}}, c_{\text{ver}}\} > 0$. Letting $t \to \infty$ and using $0 \leq W(P_t) \leq J^\star$ gives $W(P) \leq \mathbb{E}_\pi[C_T]$. Thus no pool-based policy beats $W(P)$, and the policy defining $W(P)$ attains it.

Conclusion. For $P = \emptyset$, (4) reduces to $W(\emptyset) = J^\star$. Hence an optimal pool-based policy can be implemented by generating a single fresh score r, verifying it when $h(r) > \tau^\star$, discarding it when $h(r) < \tau^\star$, and breaking ties arbitrarily at $h(r) = \tau^\star$. This is exactly the streaming policy of Lemma C.1. □

D Appendix of Section 4

Proof of Theorem 4.2. Fix a feasible prompt x, and suppress the dependence on x. Let $(h^\star, \mathcal{D})$ be the induced active-search instance. Soundness is immediate from the definition of $\mathcal{A}$: the algorithm returns only after querying the verifier and observing label 1.

We now prove the expected-cost bound.

Step 1: Reduce the oracle to reward thresholds. Since $h^\star$ is non-decreasing, Lemma D.1 implies that the distribution-aware optimum is exactly the infimum over reward-threshold policies:

$$J^\star(h^\star, \mathcal{D}) = \inf_{t:\, s_t > 0} J_t.$$

Step 2: Compare ADAP to any fixed reward threshold. By Lemma D.8, for every threshold t with $s_t > 0$,

$$\mathbb{E}[J(\mathcal{A}; h^\star, \mathcal{D})] \leq 154 J_t.$$

Step 3: Optimize over the threshold. Since the previous inequality holds for every threshold t with $s_t > 0$, we may take the infimum over such thresholds:

$$\mathbb{E}[J(\mathcal{A}; h^\star, \mathcal{D})] \le 154 \inf_{t:\, s_t > 0} J_t.$$

Using Lemma D.1 again,

$$\mathbb{E}[J(\mathcal{A}; h^\star, \mathcal{D})] \le 154 J^\star(h^\star, \mathcal{D}).$$

This proves the theorem. □

Lemma D.1 (Monotone oracle reduces to reward thresholds)**.** *Fix an active search instance $(h^\star, \mathcal{D})$ with $R \sim \mathcal{D}$ and $\Pr(V = 1) > 0$. Assume that $h^\star$ is non-decreasing. For each threshold $t \in \mathbb{R}$, define*

$$q_t := \Pr_{R\sim\mathcal{D}}(R \ge t), \qquad s_t := \mathbb{E}_{R\sim\mathcal{D}}\left[h^\star(R)\mathbf{1}\{R \ge t\}\right],$$

and, whenever $s_t > 0$,

$$J_t := \frac{c_{rew} + c_{ver} q_t}{s_t}.$$

Then the distribution-aware optimum is the infimum over reward-threshold policies:

$$J^\star(h^\star, \mathcal{D}) = \inf_{t:\, s_t > 0} J_t.$$

Proof. By Lemmas C.1 and C.2, an optimal distribution-aware policy is a (streaming) threshold rule in the success probability: there is a break-even $\tau^\star$ such that the policy verifies a candidate of score r iff $h^\star(r) > \tau^\star$, with arbitrary tie-breaking when $h^\star(r) = \tau^\star$. Since $h^\star$ is non-decreasing, the set $U^\star := \{r \in \mathbb{R} : h^\star(r) > \tau^\star\}$ is an *upper set*.

It is convenient to analyze policies of the following form. For any measurable $U \subseteq \mathbb{R}$, consider the policy that repeatedly generates $R \sim \mathcal{D}$, pays c_{rew}, verifies iff $R \in U$ (paying c_{ver}), and stops upon the first positive verifier label. Let $q(U) := \Pr(R \in U)$ and $s(U) := \mathbb{E}[h^\star(R)\mathbf{1}\{R \in U\}]$. Writing $J(U)$ for the expected cost of this policy, we have the one-step recursion

$$J(U) = c_{\text{rew}} + c_{\text{ver}}\, q(U) + (1 - s(U))\, J(U),$$

since each round incurs expected cost $c_{\text{rew}} + c_{\text{ver}} q(U)$ and succeeds with probability $s(U)$. Solving gives

$$J(U) = \frac{c_{\text{rew}} + c_{\text{ver}}\, q(U)}{s(U)} \qquad \text{whenever } s(U) > 0.$$

Now specialize to $U^\star$. Define $q_\star := q(U^\star) = \Pr(R \in U^\star)$ and $s_\star := s(U^\star) = \mathbb{E}[h^\star(R)\mathbf{1}\{R \in U^\star\}]$. By Lemma C.1, $\tau^\star$ satisfies the break-even identity $c_{\text{ver}}\, \mathbb{E}[(h^\star(R) - \tau^\star)_+] = \tau^\star c_{\text{rew}}$ and the optimal value is $J^\star(h^\star, \mathcal{D}) = c_{\text{ver}}/\tau^\star$. Moreover, since $U^\star = \{r : h^\star(r) > \tau^\star\}$, we can rewrite

$$(h^\star(R) - \tau^\star)_+ = (h^\star(R) - \tau^\star)\mathbf{1}\{R \in U^\star\},$$

so $\mathbb{E}[(h^\star(R) - \tau^\star)_+] = s_\star - \tau^\star q_\star$. Plugging into the break-even identity yields $c_{\text{ver}}(s_\star - \tau^\star q_\star) = \tau^\star c_{\text{rew}}$, i.e.,

$$c_{\text{ver}} s_\star = \tau^\star(c_{\text{rew}} + c_{\text{ver}} q_\star).$$

Dividing by $\tau^\star s_\star$ gives

$$\frac{c_{\text{rew}} + c_{\text{ver}} q_\star}{s_\star} = \frac{c_{\text{ver}}}{\tau^\star} = J^\star(h^\star, \mathcal{D}).$$

In other words, the upper-set policy associated with $U^\star$ attains the distribution-aware optimum.

It remains to relate this upper-set policy to *ordinary reward thresholds*. Since $U^\star$ is an upper subset of $\mathbb{R}$, there exists a sequence of thresholds $(t_n)_{n\ge1}$ such that $\{r : r \ge t_n\} \uparrow U^\star$ (e.g., if $U^\star = (t_0, \infty)$ take $t_n = t_0 + 1/n$, and if $U^\star = [t_0, \infty)$ take $t_n = t_0$). Let $q_n := \Pr(R \ge t_n)$ and $s_n := \mathbb{E}[h^\star(R)\mathbf{1}\{R \ge t_n\}]$. By monotone convergence, $q_n \to q_\star$ and $s_n \to s_\star$. Also $s_\star > 0$: otherwise $\mathbb{E}[(h^\star(R) - \tau^\star)_+] = 0$, contradicting $\tau^\star c_{\text{rew}} > 0$. Hence $s_n > 0$ for all large n, and for those n,

$$J_{t_n} = \frac{c_{\text{rew}} + c_{\text{ver}} q_n}{s_n} \longrightarrow \frac{c_{\text{rew}} + c_{\text{ver}} q_\star}{s_\star} = J^\star(h^\star, \mathcal{D}).$$

Therefore $\inf_{t:\, s_t > 0} J_t \le J^\star(h^\star, \mathcal{D})$.

For the reverse inequality, fix any threshold t with $s_t > 0$. The policy that generates i.i.d. and verifies iff $R \geq t$ is a valid distribution-aware policy with expected cost J_t, so by definition of $J^\star$ we have $J^\star(h^\star, \mathcal{D}) \leq J_t$ for every such t. Taking the infimum over t gives $J^\star(h^\star, \mathcal{D}) \leq \inf_{t:\, s_t>0} J_t$. Combining both directions completes the proof. □

Lemma D.2. *Let $q_t = \Pr(R \geq t)$ and $s_t = \Pr(R \geq t, V = 1)$. For every threshold $t \in \mathbb{R}$ with $q_t > 0$ and $s_t > 0$, there are integers $a_t, b_t \geq 0$ such that*

$$2^{-a_t-1} < q_t \leq 2^{-a_t}, \qquad 2^{-b_t} \leq s_t < 2^{-b_t+1}.$$

Moreover $a_t \leq b_t$, and $J_t \leq B_{a_t,b_t} \leq 4J_t$ where $B_{a_t,b_t} = c_{rew}2^{b_t} + c_{ver}2^{b_t-a_t}$.

Proof. The dyadic indices exist by construction. Since $s_t \leq q_t$, we must have $a_t \leq b_t$. For the first inequality, $s_t \geq 2^{-b_t}$ and $q_t \leq 2^{-a_t}$ imply $J_t = \frac{c_{\text{rew}}+c_{\text{ver}}q_t}{s_t} \leq (c_{\text{rew}} + c_{\text{ver}}2^{-a_t})2^{b_t} = B_{a_t,b_t}$. For the reverse inequality, $s_t < 2^{-b_t+1}$ gives $2^{b_t} < 2/s_t$, while $q_t > 2^{-a_t-1}$ gives $2^{-a_t} < 2q_t$. Therefore $B_{a_t,b_t} = (c_{\text{rew}} + c_{\text{ver}}2^{-a_t})2^{b_t} < (c_{\text{rew}} + 2c_{\text{ver}}q_t)\frac{2}{s_t} \leq 4J_t$. □

Lemma D.3. *Assume $h^\star$ is non-decreasing. Let $A \subseteq B$ be finite multisets of reward scores. Fix integers k, ℓ such that $k \leq |A|$, and $k \leq \ell \leq |B|$. Let A_k be the multiset of the k largest elements of A, and let B_ℓ be the multiset of the ℓ largest elements of B. Then*

$$1 - \prod_{r\in B_\ell} \big(1 - h^\star(r)\big) \geq 1 - \prod_{r\in A_k} \big(1 - h^\star(r)\big).$$

Proof. Write the elements of A and B in non-increasing order:

$$a_{(1)} \geq a_{(2)} \geq \cdots \geq a_{(|A|)}, \qquad b_{(1)} \geq b_{(2)} \geq \cdots \geq b_{(|B|)}.$$

Since $A \subseteq B$, for every $j \leq |A|$, we have $b_{(j)} \geq a_{(j)}$. In particular, this holds for every $j \leq k$. Since $h^\star$ is non-decreasing, $h^\star(b_{(j)}) \geq h^\star(a_{(j)})$ for all $j \leq k$. Therefore, $1 - h^\star(b_{(j)}) \leq 1 - h^\star(a_{(j)})$ for all $j \leq k$. Multiplying over $j = 1, \ldots, k$, we get

$$\prod_{j=1}^{k}\big(1 - h^\star(b_{(j)})\big) \leq \prod_{j=1}^{k}\big(1 - h^\star(a_{(j)})\big).$$

Because $k \leq \ell$, the product over the top ℓ elements of B contains these first k factors and additional factors in $[0, 1]$. Hence

$$\prod_{j=1}^{\ell}\big(1 - h^\star(b_{(j)})\big) \leq \prod_{j=1}^{k}\big(1 - h^\star(b_{(j)})\big) \leq \prod_{j=1}^{k}\big(1 - h^\star(a_{(j)})\big).$$

Taking complements concludes the claim. □

Lemma D.4. *Fix a threshold t with dyadic pair (a, b), and let $s_\star$ be the shell such that $(a, b) \in S_{s_\star}$. Then for every integer $u \geq 0$, $(a, b + u) \in S_{s_\star+u}$. Consequently, the stage in shell $s_\star + u$ has*

$$m_{s_\star+u} \geq \lceil 2^{b+u+1} \rceil, \qquad k_{s_\star+u} \geq \lceil 6 \cdot 2^{b-a+u} \rceil.$$

Proof. We have

$$B_{a,b+u} = c_{\text{rew}}2^{b+u} + c_{\text{ver}}2^{b+u-a} = 2^u B_{a,b}.$$

Since $(a, b) \in S_{s_\star}$, $2^{s_\star}c_{\min} \leq B_{a,b} < 2^{s_\star+1}c_{\min}$. Multiplying by 2^u gives $2^{s_\star+u}c_{\min} \leq B_{a,b+u} < 2^{s_\star+u+1}c_{\min}$. Hence $(a, b + u) \in S_{s_\star+u}$. Since this pair is in the shell, the definitions of $b^\star_{s_\star+u}$ and $j^\star_{s_\star+u}$ imply $b^\star_{s_\star+u} \geq b + u$ and $j^\star_{s_\star+u} \geq b - a + u$. The bounds on $m_{s_\star+u}$ and $k_{s_\star+u}$ follow. □

Lemma D.5. *Fix a threshold t with dyadic pair (a, b), and let $s_\star$ be the matching shell. For every $u \geq 0$, conditional on the algorithm reaching shell $s_\star + u$, the probability that shell $s_\star + u$ fails to output a verified positive is at most*

$$\rho_u = e^{-2^{u+1}} + \tfrac{1}{3}e^{-\beta 2^{u+1}}, \qquad \beta := 3\ln 3 - 2 > 1.$$

Proof. By Lemma D.4, shell $s_\star + u$ generates at least $m_u := 2^{b+u+1}$ fresh samples and has verification budget at least $k_u := 6 \cdot 2^{b-a+u}$. Condition on the history at the start of shell $s_\star + u$. The fresh samples generated in this shell are independent of this history. We analyze any fixed subset of m_u fresh samples generated in the shell.

Let

$$N_t := \sum_{i=1}^{m_u} \mathbf{1}\{R_i \geq t\}, \quad \text{and} \quad Z_t := \sum_{i=1}^{m_u} \mathbf{1}\{R_i \geq t, V_i = 1\}.$$

If $Z_t \geq 1$ and $N_t \leq k_u$, then the top k_u rewards among these m_u fresh samples contain a verified-positive candidate. Since these fresh samples are part of the pool and since the actual verification budget in the shell is at least k_u, Lemma D.3 implies that the conditional success probability of the actual pool-verification step is at least the conditional probability of success from these m_u fresh samples alone.

It remains to bound the probability that the event $Z_t \geq 1$ and $N_t \leq k_u$ fails. Since $Z_t \sim \mathrm{Bin}(m_u, s_t)$ and $s_t \geq 2^{-b}$, we have $m_u s_t \geq 2^{b+u+1} 2^{-b} = 2^{u+1}$. Thus

$$\Pr(Z_t = 0) \leq e^{-m_u s_t} \leq e^{-2^{u+1}}.$$

Next, $N_t \sim \mathrm{Bin}(m_u, q_t)$ and $q_t \leq 2^{-a}$. Hence

$$\mathbb{E}[N_t] \leq 2^{b+u+1} 2^{-a} = 2^{b-a+u+1},$$

Let $\mu := 2^{b-a+u+1}$, so $\mathbb{E}[N_t] \leq \mu$ and $k_u = 3\mu$. Each indicator $\mathbf{1}\{R_i \geq t\}$ contributes the expected factor $(1 - q_t)3^0 + q_t 3^1 = 1 + 2q_t$. Independence therefore gives

$$\mathbb{E}[3^{N_t}] = (1 + 2q_t)^{m_u} \leq e^{2m_u q_t} \leq e^{2\mu},$$

where the inequalities use $1 + x \leq e^x$ and $m_u q_t \leq \mu$. Thus

$$\begin{aligned} \Pr(N_t > k_u) &= \Pr(N_t \geq 3\mu + 1) = \Pr(3^{N_t} \geq 3^{3\mu+1}) \\ &\leq \frac{\mathbb{E}[3^{N_t}]}{3^{3\mu+1}} \leq \frac{e^{2\mu}}{3^{3\mu+1}} \\ &= \frac{1}{3} e^{-\beta\mu} \leq \frac{1}{3} e^{-\beta 2^{u+1}}. \end{aligned}$$

The first equality uses that N_t and $k_u = 3\mu$ are integers; the second uses that $x \mapsto 3^x$ is increasing. The first inequality is Markov's inequality applied to the nonnegative 3^{N_t} (the exponential Chernoff step), and the second uses the moment bound above. The last equality uses $\beta = 3 \ln 3 - 2$, and the last inequality uses $\mu \geq 2^{u+1}$, since $b \geq a$. A union bound gives the claim. □

Lemma D.6. *For every nonempty shell S_s, the deterministic cost $\Lambda_s := c_{rew} m_s + c_{ver} k_s$ of shell s satisfies $\Lambda_s \leq 16 \cdot 2^s c_{\min}$.*

Proof. Substituting definitions of m_s and k_s from Algorithm 1, we have

$$\Lambda_s = c_{\text{rew}}(2^{b_s^\star + 1}) + c_{\text{ver}}(6 \cdot 2^{j_s^\star}) \leq 2c_{\text{rew}} 2^{b_s^\star} + 6c_{\text{ver}} 2^{j_s^\star}.$$

Since $b_s^\star$ is attained by some pair in S_s, $c_{\text{rew}} 2^{b_s^\star} < 2^{s+1} c_{\min}$. Similarly, since $j_s^\star$ is attained by some pair in S_s, $c_{\text{ver}} 2^{j_s^\star} < 2^{s+1} c_{\min}$. Therefore $\Lambda_s < 2 \cdot 2^{s+1} c_{\min} + 6 \cdot 2^{s+1} c_{\min} = 16 \cdot 2^s c_{\min}$. □

Lemma D.7. *Let $\beta := 3 \ln 3 - 2$ and*

$$\Gamma = 1 + \sum_{u \geq 1} 2^u \left(e^{-2^u} + \tfrac{1}{3} e^{-\beta 2^u} \right).$$

Then $\Gamma < 1.405$.

Proof. For $a \geq 1$, define $g(a) := \sum_{u \geq 1} 2^u e^{-a 2^u}$. Then

$$\Gamma = 1 + g(1) + \frac{1}{3} g(\beta).$$

We first bound $g(a)$. Keep the first term $2e^{-2a}$ and bound all remaining terms. Write $u = j + 2$, where $j \geq 0$ is an integer. Since $2^j \geq j + 1$,

$$\begin{aligned} 2^{j+2}e^{-a2^{j+2}} &= 4 \cdot 2^j e^{-4a2^j} \\ &\leq 4 \cdot 2^j e^{-4a(j+1)} = 4e^{-4a}(2e^{-4a})^j. \end{aligned}$$

Thus each remaining term is bounded by the corresponding term of a geometric series with first term $4e^{-4a}$ and ratio $2e^{-4a} < 1$. Summing these bounds gives

$$g(a) \leq 2e^{-2a} + 4e^{-4a} \sum_{j \geq 0} (2e^{-4a})^j = 2e^{-2a} + \frac{4e^{-4a}}{1 - 2e^{-4a}}.$$

Applying this bound at $a = 1$ and $a = \beta > 1$, we obtain

$$\begin{aligned} \Gamma &= 1 + g(1) + \frac{1}{3} g(\beta) \\ &\leq 1 + 2e^{-2} + \frac{4e^{-4}}{1 - 2e^{-4}} + \frac{1}{3}\left(2e^{-2\beta} + \frac{4e^{-4\beta}}{1 - 2e^{-4\beta}}\right) \\ &< 1.40425 < 1.405. \end{aligned}$$

The numerical bound follows by substituting $e^{-2} < 0.13534$ and $e^{-2\beta} = e^4/729 < 0.07492$; each summand is increasing in the corresponding exponential term.

□

Lemma D.8. *For every threshold t with $s_t > 0$, the expected cost of the shellwise algorithm is at most $154 J_t$.*

Proof. Let (a, b) be the dyadic pair for t, and let $s_\star$ be the shell with $(a, b) \in S_{s_\star}$. By Lemma D.2, $2^{s_\star} c_{\min} \leq B_{a,b} \leq 4 J_t$.

First consider the deterministic cost before shell $s_\star$. By Lemma D.6,

$$\sum_{s=0}^{s_\star - 1} \Lambda_s \leq 16 c_{\min} \sum_{s=0}^{s_\star - 1} 2^s < 16 \cdot 2^{s_\star} c_{\min}.$$

Now consider the cost from shell $s_\star$ onward. The algorithm reaches shell $s_\star$ with probability at most one. For $u \geq 1$, in order to reach shell $s_\star + u$, the algorithm must have reached and failed shell $s_\star + u - 1$. By Lemma D.5, this conditional failure probability is at most ρ_{u-1}. Therefore

$$\mathbb{E}[\text{cost from shell } s_\star \text{ onward}] \leq 16 \cdot 2^{s_\star} c_{\min} \left(1 + \sum_{u \geq 1} 2^u \rho_{u-1}\right).$$

Since $\rho_{u-1} = e^{-2^u} + \frac{1}{3} e^{-\beta 2^u}$, Lemma D.7 gives $\Gamma = 1 + \sum_{u \geq 1} 2^u \rho_{u-1} < 1.405$, and hence

$$\mathbb{E}[\text{cost from shell } s_\star \text{ onward}] \leq 16\Gamma \cdot 2^{s_\star} c_{\min}.$$

Combining the two parts,

$$\begin{aligned} \mathbb{E}[J(\mathcal{A}; h, D)] &\leq 16(1 + \Gamma) \cdot 2^{s_\star} c_{\min} \leq 16(1 + \Gamma) B_{a,b} \\ &\leq 64(1 + \Gamma) J_t < 153.92 J_t < 154 J_t. \end{aligned}$$

□

E Appendix of Section 6

Proof. Fix an integer $m \leq \mathfrak{s}_0$. By Definition 6.1, choose $x_1, \ldots, x_m \in \mathcal{X}$ and $h_1, \ldots, h_m \in \mathcal{H}$ with $h_i(x_j) = \mathbb{1}[i = j]$. Let $\mathcal{D}$ be uniform on $\{x_1, \ldots, x_m\}$.

Oracle upper bound. For any target h_i, the distribution-aware oracle generates i.i.d. samples from $\mathcal{D}$ until x_i appears, then verifies it once. Since $\mathcal{D}(x_i) = 1/m$,

$$J^\star(h_i, \mathcal{D}) \;\leq\; c_{\text{rew}}\, m + c_{\text{ver}} \qquad \text{for every } i \in [m]. \tag{6}$$

Algorithm lower bound. Place a uniform prior $I \sim \mathrm{Unif}([m])$ on the target index, and let T be the first verifier call that returns label 1. We think of each verifier call as the algorithm guessing an index in $[m]$: the response is 1 if the guess equals I and 0 otherwise.

Suppose the algorithm has made q guesses, all returning 0 (i.e. $T > q$). It has ruled out at most q indices, so the posterior on I is uniform over the remaining set, which has size at least $m - q$. Whatever index the algorithm guesses next is a deterministic function of its history, so it hits I with probability at most $\frac{1}{m-q}$. Hence

$$\Pr(T > q+1 \mid T > q) \;\geq\; 1 - \frac{1}{m-q}.$$

Multiplying the survival probabilities,

$$\Pr(T > q) \;\geq\; \prod_{k=0}^{q-1}\left(1 - \frac{1}{m-k}\right) \;=\; \prod_{k=0}^{q-1}\frac{m-k-1}{m-k} \;=\; \frac{m-q}{m},$$

where the product telescopes (each numerator cancels the next denominator). Using $\mathbb{E}[T] = \sum_{q\geq 0}\Pr(T > q)$,

$$\mathbb{E}[T] \;\geq\; \sum_{q=0}^{m-1}\frac{m-q}{m} \;=\; \frac{m+1}{2} \;\geq\; \frac{m}{2}.$$

The algorithm pays at least c_{ver} per verifier call, so $\mathbb{E}[J(\mathcal{A}; h_I, \mathcal{D})] \geq c_{\text{ver}}\,\mathbb{E}[T] \geq \frac{c_{\text{ver}}}{2}m$, where the expectation is over I, the samples, and $\mathcal{A}$'s randomness. Averaging over I produces some $i^\star \in [m]$ with

$$\mathbb{E}[J(\mathcal{A}; h_{i^\star}, \mathcal{D})] \;\geq\; \tfrac{c_{\text{ver}}}{2}\,m. \tag{7}$$

Set $h^\star = h_{i^\star}$ and $\gamma := c_{\text{ver}}/c_{\text{rew}}$. Combining (6) and (7),

$$\frac{\mathbb{E}[J(\mathcal{A}; h^\star, \mathcal{D})]}{J^\star(h^\star, \mathcal{D})} \;\geq\; \frac{(c_{\text{ver}}/2)\,m}{c_{\text{rew}}\,m + c_{\text{ver}}} \;=\; \frac{1}{2}\cdot\frac{\gamma m}{m+\gamma} \;\geq\; \frac{1}{4}\min\{m, \gamma\}.$$

The bound holds for every $m \leq \mathfrak{s}_0$ yields the claimed $\frac{1}{4}\min\{\mathfrak{s}_0, \gamma\}$. □

E.1 Extension to Probabilistic Concept Classes

Next we give an extension of the centered star number Definition 6.1 for probabilistic concept classes.

Definition E.1. For a probabilistic concept class $\mathcal{H} \subseteq [0,1]^{\mathcal{X}}$ and $\eta \in (0,1]$, define η-centered star number of $\mathcal{H}$ as

$$\mathfrak{s}_{0,\eta} := \sup\left\{m : \exists x_1, \ldots, x_m,\; h_1, \ldots, h_m \in \mathcal{H} \text{ such that } h_i(x_i) \geq \eta,\; h_i(x_j) = 0\;\forall j \neq i\right\}.$$

The following corollary extends Theorem 6.2 to the probabilistic concept classes.

Corollary E.2. *For every integer $m \leq \mathfrak{s}_{0,\eta}$, every sound active search algorithm has a worst-case instance $(h^\star, \mathcal{D})$ such that* $\dfrac{\mathbb{E}[J(\mathcal{A}; h^\star, \mathcal{D})]}{J^\star(h^\star, \mathcal{D})} \geq \dfrac{\eta}{4}\min\left\{\mathfrak{s}_{0,\eta}, \dfrac{c_{ver}}{c_{rew}}\right\}.$

E.2 Connection Between Active Learning and Active Search

This section formalizes the relation between pool-based active learning and pool-based active search. Active learning aims to output a classifier with small error, while active search only aims to find one positive example. We show that, on any instance with sufficiently large positive mass, active learning implies active search.

For a distribution $D \in \Delta(\mathcal{X})$ and two measurable functions $f, g : \mathcal{X} \to \{0,1\}$, define

$$\mathrm{err}_D(f, g) := \Pr_{X\sim D}(f(X) \neq g(X)).$$

We use this both for the error of a learner's output relative to a target and for the disagreement between two concepts. In particular, if $h_0 \equiv 0$, then

$$\mathrm{err}_D(h, h_0) = \Pr_{X\sim D}(h(X) = 1).$$

Definition E.3. An (n, m) pool-based active learning algorithm $\mathcal{A}$ proceeds as follows. First, it receives an unlabeled sample

$$S_n = (X_1, \ldots, X_n) \sim D^n.$$

Then, for rounds $t = 1, \ldots, m$, the algorithm adaptively chooses an index $I_t \in [n]$ as a function of S_n and the previously observed labeled examples

$$(X_{I_1}, Y_{I_1}), \ldots, (X_{I_{t-1}}, Y_{I_{t-1}}).$$

After at most m label queries, the algorithm outputs a classifier $\widehat{h} : \mathcal{X} \to \{0, 1\}$.

Definition E.4. Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ be a binary concept class, and fix $\varepsilon, \delta \in [0, 1]$. We say that $\mathcal{A}$ (ε, δ)-learns $\mathcal{H}$ with unlabeled sample complexity n and label query complexity m if, for every $h^\star \in \mathcal{H}$ and every $D \in \Delta(\mathcal{X})$,

$$\Pr\left(\mathrm{err}_D(\widehat{h}, h^\star) \le \varepsilon\right) \ge 1 - \delta,$$

where the probability is over $S_n \sim D^n$ and any internal randomness of $\mathcal{A}$.

Definition E.5 (Pool-based active search)**.** Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ be a binary concept class, and fix $\delta \in [0, 1]$. We say that an active search algorithm $\mathcal{A}$ finds a positive example for $\mathcal{H}$ with unlabeled sample complexity n and label query complexity m with probability at least $1 - \delta$ if, for every $h^\star \in \mathcal{H}$ and every $D \in \Delta(\mathcal{X})$ with $\Pr_{X\sim D}(h^\star(X) = 1) > 0$,

$$\Pr\big(h^\star(X_{\widehat{I}}) = 1\big) \ge 1 - \delta,$$

where $\widehat{I} \in [n]$ is the index output by $\mathcal{A}$, and the probability is over $S_n \sim D^n$ and any internal randomness of $\mathcal{A}$.

Theorem E.6 (Active learning implies active search)**.** *Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{X}}$ be a binary concept class containing the all-zero concept $h_0 \equiv 0$. Suppose that $\mathcal{A}_{\mathrm{learn}}$ is an (n, m) pool-based active learning algorithm that (ε, δ)-learns $\mathcal{H}$. Then there exists an (n, m) pool-based active search algorithm $\mathcal{A}_{\mathrm{search}}$ such that, for every $h^\star \in \mathcal{H}$ and every $D \in \Delta(\mathcal{X})$ with*

$$p_\star := \Pr_{X\sim D}(h^\star(X) = 1) > 2\varepsilon,$$

we have

$$\Pr_{h^\star}\big(h^\star(X_{\widehat{I}}) = 1\big) \ge 1 - 2\delta.$$

Proof. The algorithm $\mathcal{A}_{\mathrm{search}}$ simulates $\mathcal{A}_{\mathrm{learn}}$ on the pool $S_n \sim D^n$, forwards each label query to the $h^\star$-oracle, and outputs the first queried index whose label is 1. If no such index is queried, it outputs an arbitrary index. Let E be the event that no queried label is 1, and let $\mathbb{P}_h$ denote the law of the execution under target h.

First, the output $\widehat{h}$ of $\mathcal{A}_{\mathrm{learn}}$ distinguishes h_0 from $h^\star$. Define

$$T := \mathbb{1}\left[\mathrm{err}_D(\widehat{h}, h_0) \le \varepsilon\right].$$

Under h_0, the learning guarantee gives $\mathbb{P}_{h_0}(T = 1) \ge 1 - \delta$. Under $h^\star$, since

$$p_\star = \mathrm{err}_D(h^\star, h_0) > 2\varepsilon,$$

the triangle inequality implies that $T = 1$ forces

$$\mathrm{err}_D(\widehat{h}, h^\star) \ge p_\star - \mathrm{err}_D(\widehat{h}, h_0) > \varepsilon.$$

Thus $\mathbb{P}_{h^\star}(T = 1) \le \delta$. By the variational characterization of total variation,

$$\|\mathbb{P}_{h_0} - \mathbb{P}_{h^\star}\|_{\mathrm{TV}} \ge \mathbb{P}_{h_0}(T = 1) - \mathbb{P}_{h^\star}(T = 1) \ge 1 - 2\delta.$$

We now upper bound the same total variation by the probability that $\mathcal{A}_{\mathrm{learn}}$ queries a positive point under $h^\star$. Couple the two executions using the same pool S_n and the same internal randomness. If the $h^\star$-execution never observes label 1, then all observed labels are identical to those under h_0. Since the next query is a function only of the pool, the randomness, and the previously observed labels, the two executions are identical on E. Hence

$$\|\mathbb{P}_{h_0} - \mathbb{P}_{h^\star}\|_{\mathrm{TV}} \le \mathbb{P}_{h^\star}(E^c).$$

Combining the two bounds gives

$$\mathbb{P}_{h^\star}(E^c) \geq 1 - 2\delta.$$

On E^c, the simulated learner queries at least one point with label 1, and $\mathcal{A}_{\text{search}}$ outputs such an index. Therefore

$$\Pr_{h^\star}\big(h^\star(X_{\widehat{I}}) = 1\big) \geq 1 - 2\delta.$$

□

Remark E.7 (Strong separation). Theorem E.6 has no converse: there are instances on which active search needs one label while active learning requires $\Omega(d)$.

Construction. Let $\mathcal{Z} = \{c, e_1, \ldots, e_d\} \subset \mathbb{R}^d$ with $c = (1, \ldots, 1)$ and e_j the j-th basis vector, and let $\mathcal{D}(c) = \frac{1}{2}$, $\mathcal{D}(e_j) = \frac{1}{2d}$. For $w \in \{0, 1, 2\}^d$, define

$$h_w(z) := \mathbb{1}\left[w^\top z \geq \tfrac{3}{2}\right], \qquad \mathcal{H} := \{h_w : w \in \{0, 1, 2\}^d\}.$$

Every h_w is monotone in coordinates with nonnegative weights, and $\mathcal{H}$ contains $h_0 \equiv 0$ (take $w = 0$). For $w \in \{1, 2\}^d$, identifying $S(w) := \{j : w_j = 2\}$ gives $h_w(c) = 1$ and $h_w(e_j) = \mathbb{1}[j \in S(w)]$.

Search is easy. For every $h^\star \in \mathcal{H}$ with $h^\star \neq h_0$, we have $\mathcal{D}(\{z : h^\star(z) = 1\}) \geq \mathcal{D}(c) = \frac{1}{2}$. A pool of $n = O(\log(1/\delta))$ unlabeled draws thus contains c with probability $\geq 1 - \delta$, and a single verifier call on c certifies a positive. Hence $m = 1$.

Learning is hard. Restrict to $\{h_w : w \in \{1, 2\}^d\}$, indexed by $C \subseteq [d]$ via $w_j = 1 + \mathbb{1}[j \in C]$, and let $\mathcal{A}$ be any (n, m) pool-based active learner with output $\widehat{h}$. Draw $C \sim \mathrm{Unif}(2^{[d]})$ independently of the pool and $\mathcal{A}$'s randomness, and set $B_j := \mathbb{1}[j \in C]$, so $B_1, \ldots, B_d \sim \mathrm{Bernoulli}(\frac{1}{2})^{\otimes d}$. Under target h_C, a query on c deterministically returns 1 and a query on e_j returns B_j.

Let $\mathcal{V}$ denote $\mathcal{A}$'s view (pool, randomness, observed labels) and $T \subseteq [d]$ the random set of indices with e_j queried, so $|T| \leq m$. Conditional on $\mathcal{V}$, the unqueried bits $\{B_j : j \notin T\}$ remain $\mathrm{Bernoulli}(\frac{1}{2})$, and $\widehat{h}$ is $\mathcal{V}$-measurable; hence $\Pr(\widehat{h}(e_j) \neq B_j \mid \mathcal{V}) = \frac{1}{2}$ for every $j \notin T$. Dropping the (only-helps-the-learner) contribution from c,

$$2d \cdot \mathrm{err}_{\mathcal{D}}(\widehat{h}, h_C) \,\big|\, \mathcal{V} \;\succeq\; \mathrm{Binomial}\big(d - |T|,\, \tfrac{1}{2}\big).$$

Fix $\varepsilon < \frac{1}{4}$ and suppose $m \leq d(1 - 4\varepsilon)/2$. Then $d - |T| \geq d(1 + 4\varepsilon)/2$, and Hoeffding gives

$$\Pr\big(\mathrm{err}_{\mathcal{D}}(\widehat{h}, h_C) \leq \varepsilon \,\big|\, \mathcal{V}\big) \;\leq\; \exp\Big(-\tfrac{(1-4\varepsilon)^2}{4(1+4\varepsilon)}\, d\Big) \quad \text{a.s.}$$

The tower rule preserves the bound, and for d large enough it falls below $1/3$. By Yao's principle some target in $\mathcal{H}$ forces failure with probability $> 1/3$, so $m = \Omega(d)$ for every constant $\varepsilon < 1/4$. ◁ ◁

E.3 Proof of Theorem 6.3

Proof of Theorem 6.3. Soundness is immediate since $\mathcal{A}_{\text{CS}}$ returns only after observing verifier label 1. Let $p := \Pr_{Z \sim \mathcal{D}}(h^\star(Z) = 1) > 0$. By Lemma E.9, $J^\star(h^\star, \mathcal{D}) = c_{\text{rew}}/p + c_{\text{ver}}$.

First suppose $\mathcal{A}_{\text{CS}}$ enters the verify-all branch, i.e. $\mathfrak{s}_0 > c_{\text{ver}}/c_{\text{rew}}$. The branch succeeds with geometric success probability p, so

$$\mathbb{E}[J(\mathcal{A}_{\text{CS}}; h^\star, \mathcal{D})] = \frac{c_{\text{rew}} + c_{\text{ver}}}{p}.$$

Therefore

$$\frac{\mathbb{E}[J(\mathcal{A}_{\text{CS}}; h^\star, \mathcal{D})]}{J^\star(h^\star, \mathcal{D})} = \frac{(c_{\text{rew}} + c_{\text{ver}})/p}{c_{\text{rew}}/p + c_{\text{ver}}} = \frac{1 + c_{\text{ver}}/c_{\text{rew}}}{1 + (c_{\text{ver}}/c_{\text{rew}})p} \leq 1 + \frac{c_{\text{ver}}}{c_{\text{rew}}} \leq 2\frac{c_{\text{ver}}}{c_{\text{rew}}},$$

where the last inequality uses $c_{\text{ver}} \geq c_{\text{rew}}$. In this branch, $\min\{\mathfrak{s}_0, c_{\text{ver}}/c_{\text{rew}}\} = c_{\text{ver}}/c_{\text{rew}}$, and hence

$$\mathbb{E}[J(\mathcal{A}_{\text{CS}}; h^\star, \mathcal{D})] \leq 6 \min\left\{\mathfrak{s}_0, \frac{c_{\text{ver}}}{c_{\text{rew}}}\right\} J^\star(h^\star, \mathcal{D}).$$

Now suppose $\mathcal{A}_{\mathrm{CS}}$ enters the batch branch, i.e. $\mathfrak{s}_0 \le c_{\mathrm{ver}}/c_{\mathrm{rew}}$. Since the instance is feasible and $h_0 \in \mathcal{H}$, we have $\mathfrak{s}_0 \ge 1$. Let

$$n := \left\lceil \frac{c_{\mathrm{ver}}}{c_{\mathrm{rew}}} \mathfrak{s}_0 \right\rceil .$$

By Lemma E.8, every batch $S = (Z_1, \dots, Z_n)$ admits a hitting set $I \subseteq [n]$ with $|I| \le \mathfrak{s}_0$. If the batch contains a positive point, then $A_{h^\star}(S) \neq \emptyset$, so $I \cap A_{h^\star}(S) \neq \emptyset$, and verifying all indices in I finds a positive. Thus one batch succeeds with probability $\alpha := 1 - (1-p)^n$.

The cost of one batch is at most $B := c_{\mathrm{rew}} n + c_{\mathrm{ver}} \mathfrak{s}_0$. Since $n = \lceil (c_{\mathrm{ver}}/c_{\mathrm{rew}}) \mathfrak{s}_0 \rceil$, $\mathfrak{s}_0 \le c_{\mathrm{ver}}/c_{\mathrm{rew}}$, and $\mathfrak{s}_0 \ge 1$, we have $n \le 2(c_{\mathrm{ver}}/c_{\mathrm{rew}})\mathfrak{s}_0$. Hence $B \le 3 c_{\mathrm{ver}} \mathfrak{s}_0$. Also,

$$\frac{B}{n} = c_{\mathrm{rew}} + \frac{c_{\mathrm{ver}} \mathfrak{s}_0}{n} \le 2 c_{\mathrm{rew}}. \tag{8}$$

Let $x := np$. Since $1 - p \le e^{-p}$, we have $\alpha = 1 - (1-p)^n \ge 1 - e^{-np} = 1 - e^{-x}$. If $x \le 1$, concavity of $1 - e^{-x}$ gives $1 - e^{-x} \ge x/2$. If $x > 1$, then $1 - e^{-x} \ge 1 - e^{-1} \ge 1/2$. Hence

$$\alpha \ge \frac{1}{2} \min\{1, np\}.$$

Therefore

$$\mathbb{E}[J(\mathcal{A}_{\mathrm{CS}}; h^\star, \mathcal{D})] \le \frac{B}{\alpha} \le \frac{2B}{\min\{1, np\}}.$$

If $np \le 1$, then from Eq. (8), we have $\mathbb{E}[J(\mathcal{A}_{\mathrm{CS}}; h^\star, \mathcal{D})] \le \frac{2B}{np} = \frac{2(B/n)}{p} \le \frac{4 c_{\mathrm{rew}}}{p}$. Then, notice that $\frac{4 c_{\mathrm{rew}}}{p} \le 4 J^\star(h^\star, \mathcal{D}) \le 6 \mathfrak{s}_0 J^\star(h^\star, \mathcal{D})$, where the last inequality uses $\mathfrak{s}_0 \ge 1$. If $np > 1$, then

$$\mathbb{E}[J(\mathcal{A}_{\mathrm{CS}}; h^\star, \mathcal{D})] \le 2B \le 6 c_{\mathrm{ver}} \mathfrak{s}_0 \le 6 \mathfrak{s}_0 J^\star(h^\star, \mathcal{D}),$$

where the last step uses $J^\star(h^\star, \mathcal{D}) \ge c_{\mathrm{ver}}$. Since in the batch branch $\min\{\mathfrak{s}_0, c_{\mathrm{ver}}/c_{\mathrm{rew}}\} = \mathfrak{s}_0$, this gives

$$\mathbb{E}[J(\mathcal{A}_{\mathrm{CS}}; h^\star, \mathcal{D})] \le 6 \min \left\{ \mathfrak{s}_0, \frac{c_{\mathrm{ver}}}{c_{\mathrm{rew}}} \right\} J^\star(h^\star, \mathcal{D}).$$

Combining the two branches proves the theorem. □

Lemma E.8. *Let $\mathcal{Z}$ be an arbitrary input space, $\mathcal{H} \in \{0,1\}^{\mathcal{Z}}$, and $n \in \mathbb{N}$ where $n \ge \mathfrak{s}_0(\mathcal{H})$. Let $S \in \mathcal{Z}^n$ be a sequence of length n. For every $h \in \mathcal{H}$, define $A_h(S) = \{i \in [n] \ : \ h(Z_i) = 1\}$. Then there exists $I \subseteq [n]$ such that $I \cap A_h(S) \neq \emptyset$ for every $h \in \mathcal{H}$ with $A_h(S) \neq \emptyset$, and $|I| \le \mathfrak{s}_0$.*

Proof. Let

$$\mathcal{A}_S := \{A_h(S) : h \in \mathcal{H}, \ A_h(S) \neq \emptyset\}.$$

If $\mathcal{A}_S = \emptyset$, then $I = \emptyset$ satisfies the claim. Hence assume $\mathcal{A}_S \neq \emptyset$. Since $[n]$ hits every set in $\mathcal{A}_S$, there exists a minimum-cardinality hitting set $I \subseteq [n]$, i.e., $I \cap A \neq \emptyset$ for every $A \in \mathcal{A}_S$, and $|I|$ is minimal among all such sets.

We claim that $|I| \le \mathfrak{s}_0(\mathcal{H})$. For every $i \in I$, minimality implies that there exists $h_i \in \mathcal{H}$ such that

$$A_{h_i}(S) \cap I = \{i\}.$$

Indeed, if no such h_i existed, then every set in $\mathcal{A}_S$ hit by i would also be hit by $I \setminus \{i\}$, and every set not hit by i is already hit by $I \setminus \{i\}$. Thus $I \setminus \{i\}$ would still be a hitting set, contradicting minimality of I.

Write $I = \{i_1, \dots, i_m\}$. For each $a \in [m]$, the witness h_{i_a} satisfies $A_{h_{i_a}}(S) \cap I = \{i_a\}$. Hence $h_{i_a}(Z_{i_a}) = 1$, while $h_{i_a}(Z_{i_b}) = 0$ for every $b \neq a$. Therefore

$$h_{i_a}(Z_{i_b}) = \mathbb{1}[a = b] \qquad \forall a, b \in [m].$$

Thus the selected points $Z_{i_1}, \dots, Z_{i_m}$ and the witness concepts $h_{i_1}, \dots, h_{i_m}$ form a centered star of size m. By definition of $\mathfrak{s}_0(\mathcal{H})$, $m \le \mathfrak{s}_0(\mathcal{H})$. Since $m = |I|$, this gives $|I| \le \mathfrak{s}_0(\mathcal{H})$. Finally, because I was chosen as a hitting set, $I \cap A_h(S) \neq \emptyset$ for every $h \in \mathcal{H}$ with $A_h(S) \neq \emptyset$. □

Lemma E.9. *Fix a feasible binary active-search instance $(h^\star, \mathcal{D})$ from Definition 2.1, and let $p := \Pr_{Z\sim\mathcal{D}}(h^\star(Z) = 1) > 0$. Then*

$$J^\star(h^\star, \mathcal{D}) = \frac{c_{rew}}{p} + c_{ver}.$$

Proof. The distribution-aware oracle generates i.i.d. samples from $\mathcal{D}$ until it sees a point Z with $h^\star(Z) = 1$, verifies this point once, and stops. This gives expected cost $c_{\text{rew}}/p + c_{\text{ver}}$. Conversely, any sound policy must generate at least one positive point and must make at least one verifier call. If $T_+ := \inf\{i \geq 1 : h^\star(Z_i) = 1\}$, where $Z_i \sim \mathcal{D}$, then $\mathbb{E}[T_+] = 1/p$, and hence every sound policy has expected cost at least $c_{\text{rew}}/p + c_{\text{ver}}$. □

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The paper consists of theoritical results and empirical findings. For both type of results assumptions and results have been stated clearly in abstract, introduction and main body.

   Guidelines:

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

2. **Limitations**

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

   Answer:[Yes]

   Justification: Yes. We have a section on limitation.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [Yes]

Justification: Yes, for each theoretical result, we provide very precise proof in the appendix with full assumptions and details.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: We have included our repo as a part of our submission and also included all the important hyperparameters.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: We have provided the details in our repo submitted as a zip file. You can find a readme file there. Also, all the datasets are public and available on the internet..

Guidelines:

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

6. **Experimental setting/details**

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

Answer: [Yes]

Justification: All the details are presented in our repo.

Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [No]

Justification: Due to the computational cost of our experiments and our hardware setup, we only run our experiments once.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: We have provided our computational resources in Section B.

   Guidelines:

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

9. **Code of ethics**

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

   Answer:[Yes]

   Justification: Yes, the research follows code of ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: There is no societal impact of the current work. A potential positive impact is in decreasing the energy usage of inference of LLMs.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper releases only algorithm implementation code operating on existing public benchmarks. No new models, datasets, or scraped data are introduced, and the work poses no identified risk for misuse.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: All external assets are cited and used for research evaluation in accordance with the terms stated by their respective providers.

    - **LiveCodeBench** [Jai+24]: coding evaluation benchmark. The official repository lists the project under the MIT License.
    - **HMMT** math competition problems: publicly available problems from the Harvard–MIT Mathematics Tournament archive. We use these problems only for non-commercial research evaluation; we do not redistribute the benchmark as a standalone asset.
    - **Qwen2.5-Math-7B and Qwen2.5-14B** [Qwe24]: released under Apache 2.0.
    - **Qwen2.5-Coder-3B**: used as a base coding generator under the Qwen Research License. Our use is limited to research evaluation.

- **Qwen2.5-Math-PRM-7B**: used as a math reward model under the Qwen License Agreement. The license grants rights to use, reproduce, copy, modify, and create derivatives of the materials, subject to the stated terms.
- **DeepSeek-R1-Distill-Qwen-7B** [Dee25]: released under the MIT License; the model card notes that the distilled Qwen variants are derived from Qwen-2.5 models originally licensed under Apache 2.0.
- **CodeScaler-8B** (`LARK-Lab/CodeScaler-8B` [Zhu+26]): reward model fine-tuned from `Skywork/Skywork-Reward-V2-Qwen3-8B` on `CodeScalerPair-51K`; it is under MIT license.
- **vLLM** [Kwo+23]: inference framework released under Apache 2.0.

Guidelines:

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

13. **New assets**

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

Answer: [Yes]

Justification: We release the code implementing ADAP and all baseline algorithms (SAMPLEAWARE, DAP$_k$), including the generation, reward scoring, and evaluation pipelines. The repository includes scripts to reproduce all experiments and figures reported in the paper. No new datasets or model weights are introduced; all benchmarks and models used are publicly available.

Guidelines:

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

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

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

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects. All experiments use automated LLM inference and scoring on existing public benchmarks.

Guidelines:

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

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

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

    Answer: [N/A]

    Justification: This paper does not involve human subjects.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [Yes]

    Justification: LLM assistants were used for writing, editing, and coding purposes, which do not affect the scientific methodology or results.

    Guidelines:

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