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

Anonymous Authors[1]

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]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

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 (Google DeepMind, 2025; OpenAI, 2024), 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 (Snell et al., 2025; Damani et al., 2025; Zhai et al., 2026; Raman et al., 2025; Kalayci et al., 2025; Wan et al., 2025b; Qu, 2026).

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 Figure 1, the per-prompt optimal choice 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 (Snell et al., 2025; Damani et al., 2025; Zhai et al., 2026; Raman et al., 2025; Huang et al., 2026). 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 (Snell et al., 2025; Damani et al.,

2025). 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 our central question: 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[1].

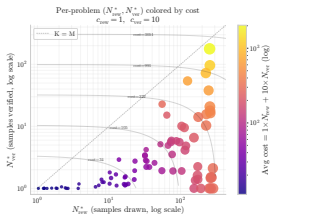


Figure 1. Per-problem optimal fixed budgets on LiveCodeBench. Color shows cost; the spread shows why one fixed generate/verify budget is inefficient.

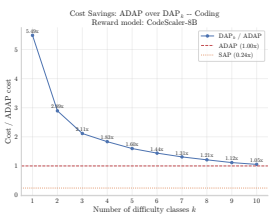


Figure 2. Cost ratios for oracle baselines relative to ADAP. Lower is cheaper; SAMPLEAWARE is a per-problem oracle and DAP$_k$ is difficulty-stratified.

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 $h_x^\star : \mathbb{R} \to [0,1]$, where $h_x^\star(r)$ 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 (Hanneke & Yang, 2015).

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

[1]Throughout the main experiments, we use $c_{\text{ver}}/c_{\text{rew}} = 10$ as the default cost ratio. In Appendix B.3, we also evaluate additional cost regimes and find that the empirical performance of our adaptive algorithm is robust across these choices.

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 84% and 88% of trials, respectively. Figure 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.

We defer a detailed discussion of related work to Appendix A in the appendix.

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 an 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* $(\mathcal{D}_x, h_x)$, 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* $\mathcal{H} \subseteq [0, 1]^{\mathbb{R}}$ (Kearns & Schapire, 1994). 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 $J^\star(h^\star, \mathcal{D}) := \inf_{\pi \in \Pi_{\text{DA}}(\mathcal{D}, h^\star)} \mathbb{E}_{\pi, h, \mathcal{D}}[c_{\text{rew}} N_{\text{rew}}(\pi) + c_{\text{ver}} N_{\text{ver}}(\pi)]$, where policies that never output a verified positive have cost $+\infty$.

We then present the main result whose proof can be found in Appendix 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 break 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_x; it only relies on the natural monotonicity assumption that higher reward scores are more likely to correspond to correct outputs. In Appendix B.2, we empirically validate this assumption on both coding and math benchmarks, both in aggregate and at the per-prompt level.

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 $\mathsf{Ver}(x, y) = 1$

$c_{\min} \leftarrow \min\{c_{\text{rew}}, c_{\text{ver}}\}$
$\mathcal{P} \leftarrow \emptyset$
for $s = 0, 1, 2, \ldots$ **do**
  $\mathcal{S}_s \leftarrow \{(a, b) \in \mathbb{Z}^2_{\geq 0} : a \leq b,$
  $2^s c_{\min} \leq c_{\text{rew}}\, 2^b + c_{\text{ver}}\, 2^{b-a} < 2^{s+1} c_{\min}\}$
  if $\mathcal{S}_s = \emptyset$ **then**
    continue
  $b_s^\star \leftarrow \max\{b : (a, b) \in \mathcal{S}_s\}$
  $j_s^\star \leftarrow \max\{b - a : (a, b) \in \mathcal{S}_s\}$
  $m_s \leftarrow \left\lceil 2^{b_s^\star + 1} \right\rceil$
  $k_s \leftarrow \left\lceil 6 \cdot 2^{j_s^\star} \right\rceil$
  Draw $y_1, \ldots, y_{m_s} \overset{\text{i.i.d.}}{\sim} \pi(\cdot \mid x)$
  Score and add $\{(y_i, \mathsf{RM}(x, y_i)) : i \in [m_s]\}$ to $\mathcal{P}$
  Relabel the elements of $\mathcal{P}$ as $(\tilde{y}_1, \tilde{R}_1), \ldots, (\tilde{y}_{|\mathcal{P}|}, \tilde{R}_{|\mathcal{P}|})$ so that $\tilde{R}_1 \geq \tilde{R}_2 \geq \cdots \geq \tilde{R}_{|\mathcal{P}|}$.
  for $j = 1, \ldots, \min\{k_s, |\mathcal{P}|\}$ **do**
    Query $\mathsf{Ver}(x, y_{(j)})$
    Remove $(\tilde{y}_j, \tilde{R}_j)$ from $\mathcal{P}$
    if $\mathsf{Ver}(x, y_{(j)}) = 1$ **then**
      return $y_{(j)}$

Assumption 4.1. *We assume $h^\star \in \mathcal{H}_{\text{non-dec}}$ where $\mathcal{H}_{\text{non-dec}} = \{h : \mathbb{R} \to [0, 1] \mid \forall\, r_1 \leq r_2\ h(r_1) \leq h(r_2)\}$ 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 Appendix D)

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), $\mathcal{A}$ is sound and satisfies $\mathbb{E}[J(\mathcal{A}_{\text{ADAP}}; h^\star, \mathcal{D})] \leq 400\, J^\star(h^\star, \mathcal{D})$, 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 Figure 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 $q_t = \Pr_{R\sim\mathcal{D}}(R \geq t)$ and $s_t = \Pr(R \geq t, V = 1) = \mathbb{E}_{R\sim\mathcal{D}}[h^\star(R)\mathbf{1}\{R \geq t\}]$. A threshold policy that generates candidates and verifies only when $R \geq t$ has expected cost $J_t = (c_{\text{rew}} + c_{\text{ver}}q_t)/s_t$. 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 $q_t \approx 2^{-a}$, $s_t \approx 2^{-b}$, and $a \leq b$, 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 (Cesa-Bianchi et al., 1997). 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 on two domains where verification is costly relative to sampling: mathematical reasoning and competitive programming.

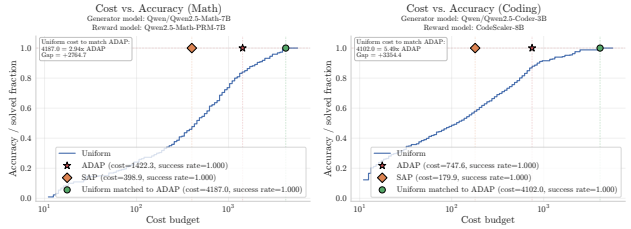
Mathematical reasoning. We use the HMMT (HMMT, 2025) February 2024 and 2025 contests (60 problems total). Solutions are sampled from `Qwen2.5-Math-7B` (base) (Qwen Team, 2024) at temperature 0.7, top-p 0.95, with $N = 512$ samples per problem. Each candidate is scored by `Qwen2.5-Math-PRM-7B` (Yang et al., 2024); 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, 22 problems remain. Results with two alternative generators (`Qwen2.5-14B` (Qwen Team, 2024) and `DeepSeek-R1-Distill-Qwen-7B` (DeepSeek-AI, 2025)) appear in Appendix B.3.1.

Coding. We use medium- and hard-difficulty problems from LiveCodeBench (LeetCode, AtCoder, Codeforces). Solutions are sampled from `Qwen2.5-Coder-3B` (base) (Hui et al., 2024) under identical settings. Each candidate is scored by `CodeScaler-8B` (Zhu et al., 2026). 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.

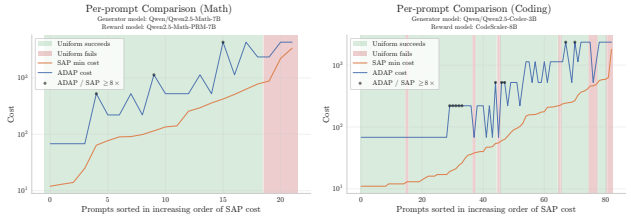
Cost model and trial protocol. We set $c_{\text{rew}} = 1$ and $c_{\text{ver}} = 10$, so the cost of N_{rew} draws and N_{ver} verifications is $N_{\text{rew}} \cdot c_{\text{rew}} + N_{\text{ver}} \cdot c_{\text{ver}}$. A trial succeeds if at least one verified sample is correct. Each (problem, policy) 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.3.2.

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. Because it observes correctness labels at decision time, it is strictly more powerful than any realizable policy, and we use its mean cost as a per-trial lower bound.



(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.

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. 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=1 applies $(N_{\text{rew}}, N_{\text{ver}})$ uniformly to all problems. As a cost-matched reference, we define UNIFORM$_{C_{\text{ADAP}}}$ as the best uniform pair whose cost does not exceed ADAP's mean cost. This is a special case of DAP$_1$ that matches ADAP's budget but does not guarantee 100% success.

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.

Math (HMMT). ADAP achieves 100% success at mean cost 1422, while SAMPLEAWARE's mean cost is 399 (0.28×). 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 84% of trials, a notable gap that confirms a uniform fixed strategy cannot replicate ADAP's reliability at equal cost. Among difficulty-stratified baselines, DAP$_1$ costs 2.94× more than ADAP, and the gap narrows as k grows: DAP$_5$ still spends 1.04× more, while DAP$_{10}$ reaches 0.82×. With only 22 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 745, with SAMPLEAWARE costing 180 (0.24×). Figure 3 again shows ADAP's per-trial cost following SAMPLEAWARE's closely. UNIFORM$_{C_{\text{ADAP}}}$ falls short at 87.8% success, a clear failure mode of uniform strategies on a heterogeneous task. Unlike the math setting, DAP$_k$ never beats ADAP even at N_{ver}=10 (1.06×), showing that the coding task's difficulty structure is harder to exploit with fixed per-class strategies and that ADAP's online adaptation is particularly valuable here.

6. Learning-Theoretic View of Active Search

Monotonicity of the score-conditioned success function gave ADAP a constant-factor guarantee. We now show that some structure is necessary. For unstructured classes, a positive response may be hidden among many locations indistinguishable without verification, while a distribution-aware oracle can focus on the right location immediately. We formalize this obstruction using the *centered star number*, a variant of the classical star number of Hanneke & Yang (2015) adapted to finding a single positive witness.

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. Define $\mathfrak{s}_0(\mathcal{H}) = \max\{m \in \mathbb{N} : \exists z_1, \ldots, z_m \in \mathcal{Z}, \exists h_1, \ldots, h_m \in \mathcal{H}$ such that $h_i(z_j) = \mathbb{1}[i = j]\ \forall i, j\}$, with $\mathfrak{s}_0(\mathcal{H}) = \infty$ if no finite maximum exists.

Theorem 6.2. *Let $\mathcal{Z}$ be an arbitrary input space, fix costs $c_{rew}, c_{ver} > 0$, and let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{Z}}$ contain the all-zero function. For every sound active-search policy $\mathcal{A}$, there exists $h^\star \in \mathcal{H}$ and $\mathcal{D} \in \Delta(\mathcal{Z})$ such that $\mathbb{E}[J(\mathcal{A}; h^\star, \mathcal{D})] \geq \frac{1}{4}\min\{\mathfrak{s}_0(\mathcal{H}), c_{ver}/c_{rew}\} J^\star(h^\star, \mathcal{D})$.*

Theorem 6.3. *Let $\mathcal{H} \subseteq \{0,1\}^{\mathcal{Z}}$ contain $h_0 \equiv 0$ and assume $c_{ver} \geq c_{rew}$. There exists a sound active-search policy $\mathcal{A}_{\text{CS}}$, specified in Algorithm 2 in the appendix, such that for every feasible binary instance $(h^\star, \mathcal{D})$ with $h^\star \in \mathcal{H}$, $\mathbb{E}[J(\mathcal{A}_{\text{CS}}; h^\star, \mathcal{D})] \leq 6\min\{\mathfrak{s}_0(\mathcal{H}), c_{ver}/c_{rew}\} J^\star(h^\star, \mathcal{D})$.*

Interpretation. Theorems 6.2 and 6.3 show that monotonicity rules out a real obstruction: without structure, a positive witness can hide among centered-star alternatives and the tight binary-class gap is $\min\{\mathfrak{s}_0(\mathcal{H}), c_{\text{ver}}/c_{\text{rew}}\}$. Active search is weaker than active learning because it stops after one positive label; see Appendix E.2.

References

Cesa-Bianchi, N., Freund, Y., Haussler, D., Helmbold, D. P., Schapire, R. E., and Warmuth, M. K. How to use expert advice. *Journal of the ACM (JACM)*, 44(3): 427–485, 1997.

Chen, B., Zhang, F., Nguyen, A., Zan, D., Lin, Z., Lou, J.-G., and Chen, W. CodeT: Code generation with generated tests. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ktrw68Cmu9c.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Damani, M., Shenfeld, I., Peng, A., Bobu, A., and Andreas, J. Learning how hard to think: Input-adaptive allocation of LM computation. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6qUUgw9bAZ.

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

Garnett, R., Krishnamurthy, Y., Xiong, X., Schneider, J. G., and Mann, R. P. Bayesian optimal active search and surveying. In *Proceedings of the 29th International Conference on Machine Learning*, pp. 843–850, 2012. URL https://icml.cc/2012/papers/618.pdf.

Google DeepMind. Gemini achieves gold-medal level at the International Collegiate Programming Contest World Finals. Blog post, 2025. URL https://deepmind.google/blog/gemini-achieves-gold-medal-level-at-the-international-collegiate-programming-contest-world-finals/.

Hanneke, S. and Yang, L. Minimax analysis of active learning. *Journal of Machine Learning Research*, 16:3487–3602, 2015. URL https://www.jmlr.org/papers/v16/hanneke15a.html.

HMMT. HMMT february 2024 and 2025 math tournaments. https://www.hmmt.org/www/archive/problems, 2025. Harvard-MIT Mathematics Tournament.

Huang, J., Ma, W., and Zhou, Z. Optimal Bayesian stopping for efficient inference of consistent LLM answers, 2026. URL https://arxiv.org/abs/2602.05395.

Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Lu, K., Dang, K., Fan, Y., Zhang, Y., Yang, A., Men, R., Huang, F., Zheng, B., Miao, Y., Quan, S., Feng, Y., Ren, X., Ren, X., Zhou, J., and Lin, J. Qwen2.5-coder technical report, 2024. URL https://arxiv.org/abs/2409.12186.

Jiang, S., Malkomes, G., Converse, G., Shofner, A., Moseley, B., and Garnett, R. Efficient nonmyopic active search. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1714–1723, 2017. URL https://proceedings.mlr.press/v70/jiang17d.html.

Jiang, S., Malkomes, G., Abbott, M., Moseley, B., and Garnett, R. Efficient nonmyopic batch active search. In *Advances in Neural Information Processing Systems*, volume 31, pp. 1099–1109, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/a7aeed74714116f3b292a982238f83d2-Abstract.html.

Jiang, S., Moseley, B., and Garnett, R. Cost effective active search. In *Advances in Neural Information Processing Systems*, volume 32, pp. 4880–4889, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/df0e09d6f25a15a815563df9827f48fa-Abstract.html.

Kalayci, Y., Raman, V., and Dughmi, S. Optimal stopping vs best-of-N for inference-time optimization, 2025. URL https://arxiv.org/abs/2510.01394.

Kearns, M. J. and Schapire, R. E. Efficient distribution-free learning of probabilistic concepts. *Journal of Computer and System Sciences*, 48(3):464–497, 1994.

Khalifa, M., Agarwal, R., Logeswaran, L., Kim, J., Peng, H., Lee, M., Lee, H., and Wang, L. Process reward models that think, 2025. URL https://arxiv.org/abs/2504.16828.

Kiyani, S., Noorani, S., Pappas, G. J., and Hassani, H. When to trust the cheap check: Weak and strong verification for reasoning, 2026. URL https://arxiv.org/abs/2602.17633.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., Hubert, T., Choy, P., de Masson d'Autume, C., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Gowal, S., Cherepanov, A., Molloy, J., Mankowitz, D. J.,

Robson, E. S., Kohli, P., de Freitas, N., Kavukcuoglu, K., and Vinyals, O. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://arxiv.org/abs/2203.07814.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Ma, Z., Zhang, X., Zhang, J., Yu, J., Luo, S., and Tang, J. 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)*, pp. 6917–6935, 2025. URL https://aclanthology.org/2025.acl-long.343/.

Ni, A., Iyer, S., Radev, D., Stoyanov, V., Yih, W.-t., Wang, S. I., and Lin, X. V. LEVER: Learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 26106–26128, 2023. URL https://proceedings.mlr.press/v202/ni23b.html.

OpenAI. Learning to reason with LLMs. Blog post, 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

Qu, S. Adaptive test-time compute allocation via learned heuristics over categorical structure, 2026. URL https://arxiv.org/abs/2602.03975.

Qwen Team. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

Raman, V., Asi, H., and Kale, S. AdaBoN: Adaptive best-of-N alignment, 2025. URL https://arxiv.org/abs/2505.12050.

Setlur, A., Nagpal, C., Fisch, A., Geng, X., Eisenstein, J., Agarwal, R., Agarwal, A., Berant, J., and Kumar, A. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=A6Y7AqlzLW.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Wan, G., Wu, Y., Wang, H., Zhao, S., Chen, J., and Li, S. Derailer-rerailer: Adaptive verification for efficient and reliable language model reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025a. URL https://aclanthology.org/2025.findings-acl.18/.

Wan, G., Xu, Z. S., Zorc, S., Baucells, M., Hu, M., Wang, H., and Li, S. BEACON: Bayesian optimal stopping for efficient LLM sampling, 2025b. URL https://arxiv.org/abs/2510.15945.

Wang, P., Li, L., Shao, Z., Xu, R., Dai, D., Li, Y., Chen, D., Wu, Y., and Sui, Z. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024. URL https://aclanthology.org/2024.acl-long.510/.

Yang, A., Zhang, B., Hui, B., Gao, B., Yu, B., Li, C., Liu, D., Tu, J., Zhou, J., Lin, J., Lu, K., Xue, M., Lin, R., Liu, T., Ren, X., and Zhang, Z. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement, 2024. URL https://arxiv.org/abs/2409.12122.

Zhai, Z., Li, B., Xiao, B., Li, M., and Wang, X. Adaptive test-time compute allocation for reasoning LLMs via constrained policy optimization, 2026. URL https://arxiv.org/abs/2604.14853.

Zhang, L., Hosseini, A., Bansal, H., Kazemi, M., Kumar, A., and Agarwal, R. Generative verifiers: Reward modeling as next-token prediction. In *International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=Ccwp4tFEtE.

Zhang, Z., Zheng, C., Wu, Y., Zhang, B., Lin, R., Yu, B., Liu, D., Zhou, J., and Lin, J. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516, 2025b. URL https://aclanthology.org/2025.findings-acl.547/.

Zhu, X., Zhou, X., Zhu, B., Hu, H., Du, M., Zhang, H., Wang, H., and Guo, Z. CodeScaler: Scaling code LLM training and test-time inference via execution-free reward models, 2026. URL https://arxiv.org/abs/2602.17684.

Contents

Table 1. Summary of all policies on Math (HMMT, 22 problems, 10 permutations) and Coding (LiveCodeBench, 83 problems, 10 permutations). Cost ratio is relative to ADAP.

	Math (HMMT)					Coding (LiveCodeBench)				
Policy	$\overline{N_{\text{rew}}}$	$\overline{N_{\text{ver}}}$	Avg. cost	Ratio	Succ.	$\overline{N_{\text{rew}}}$	$\overline{N_{\text{ver}}}$	Avg. cost	Ratio	Succ.
SAMPLEAWARE	103.7	29.5	398.9	×0.28	1.000	92.4	8.7	179.9	×0.24	1.000
ADAP	219.6	120.2	1422.3	×1.00	1.000	140.4	60.5	745.3	×1.00	1.000
UNIFORM$_{C_{\text{ADAP}}}$	182.0	124.0	1422.0	×1.00	0.841	414.0	33.0	744.0	×1.00	0.878
DAP$_1$	497.0	369.0	4187.0	×2.94	1.000	512.0	359.0	4102.0	×5.50	1.000
DAP$_2$	–	–	2164.8	×1.52	1.000	–	–	2162.1	×2.90	1.000
DAP$_3$	–	–	1792.7	×1.26	1.000	–	–	1582.1	×2.12	1.000
DAP$_5$	–	–	1477.3	×1.04	1.000	–	–	1194.3	×1.60	1.000
DAP$_{10}$	–	–	1169.4	×0.82	1.000	–	–	786.9	×1.06	1.000

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 (Cobbe et al., 2021), while process reward models and generative verifiers score intermediate or complete reasoning traces (Lightman et al., 2024; Wang et al., 2024; Setlur et al., 2025; Zhang et al., 2025a; Khalifa et al., 2025; Zhang et al., 2025b). In code generation, AlphaCode used large-scale sampling followed by execution-based filtering, clustering, and selection (Li et al., 2022); CodeT and LEVER use generated tests or execution-aware learned verifiers to rank programs (Chen et al., 2023; Ni et al., 2023); and CodeRM studies dynamic scaling of generated unit tests to improve reward quality (Ma et al., 2025). 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 (Snell et al., 2025). Some methods allocate compute across inputs using predicted reward curves (Damani et al., 2025), constrained optimization under an average budget (Zhai et al., 2026), or prompt-level Best-of-N budget allocation (Raman et al., 2025). Others adapt computation within a single input, stopping generation when further samples are unlikely to improve reward (Kalayci et al., 2025; Wan et al., 2025b), or when answer agreement is sufficiently high in self-consistency sampling (Huang et al., 2026). 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 (Kiyani et al., 2026), while Derailer–Rerailer uses a lightweight stability check to trigger more expensive reasoning only when needed (Wan et al., 2025a). Closest to our setting, Qu (2026) studies a verification-cost-limited regime and allocate 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 (Garnett et al., 2012; Jiang et al., 2017; 2018). The closest formulation to ours is cost-effective active search (Jiang et al., 2019), 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.

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 (Kwon et al., 2023) 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.

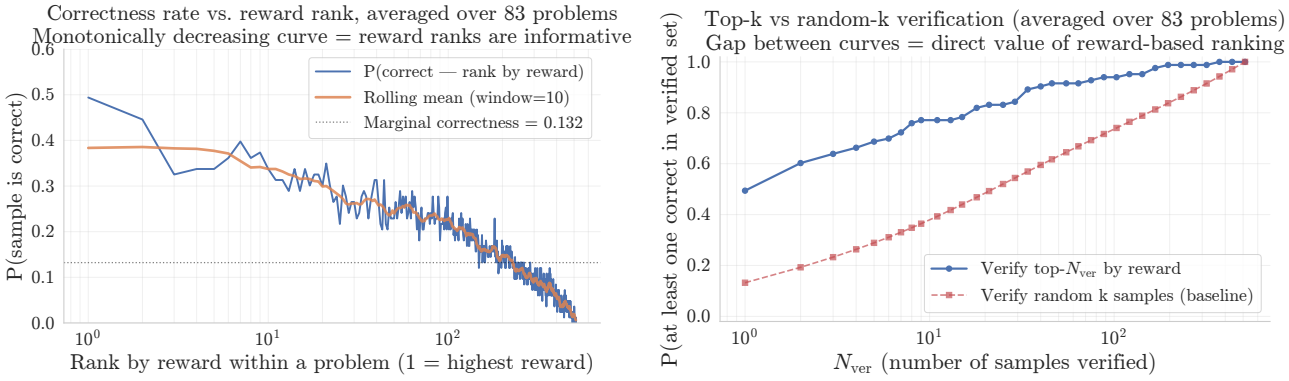
Stage	Details	Estimated cost
Math generation	3 models × 60 problems × 512 samples; 8–10 h/model on 2×L40S	∼48–60 GPU-hour
Math reward scoring	Qwen2.5-Math-PRM-7B via vLLM, 92,160 samples	∼4–6 GPU-hour
Coding generation	Qwen2.5-Coder-3B, TP=2, 329 × 512 = 168,448 samples, ∼750 tok/sample	∼26–40 GPU-hour
Coding reward scoring	CodeScaler-8B, single GPU, 42,496 samples, batch size 32	∼2–3 GPU-hour
Total GPU		**∼85–110 GPU-hour**
Coding verification	42,496 programs, 64 workers, ∼20–30 CPU-s/sample (wall time ≈4–6 h)	∼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.3.1.

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 22 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.



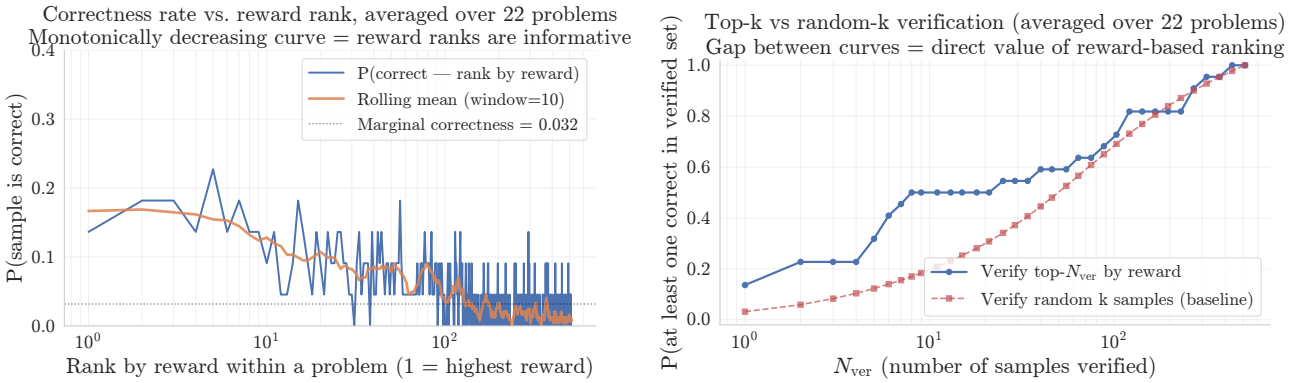
(a) Correctness probability at each reward rank.

(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).

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

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



(a) Correctness probability at each reward rank.

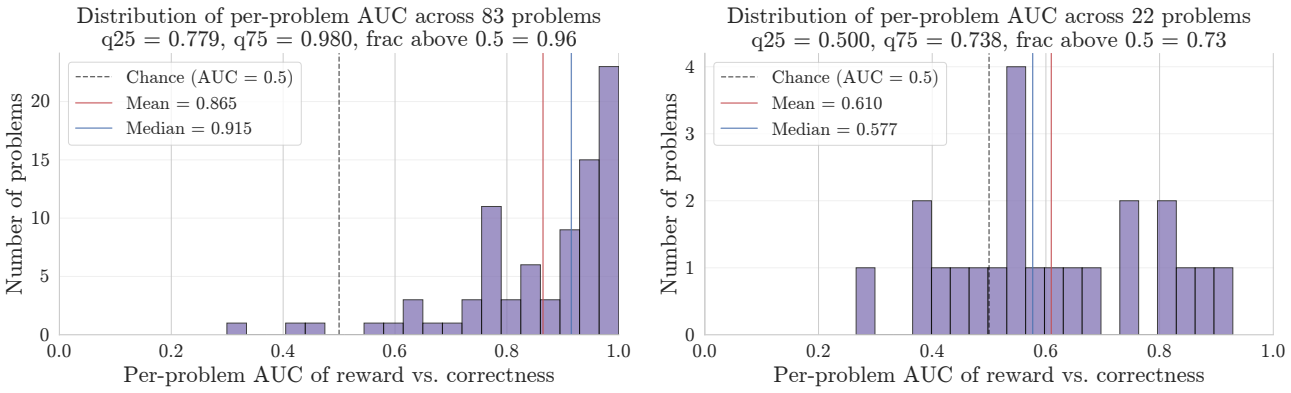
(b) Top-k vs. random-k coverage.

Figure 5. Reward signal validation on math. Left: empirical correctness probability at each reward rank, averaged over 22 problems (orange: rolling mean). Right: probability that the top-k by reward contains a correct sample (blue) vs. k uniform random samples (red dashed).

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).

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: 0.610, median: 0.577, IQR: $[0.500, 0.738]$, 73% 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 (22 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 0.610, 73% above chance).

B.3. Ablations

B.3.1. 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. $c_{\text{ver}}/c_{\text{rew}} = 10$. 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`	
	Cost	Ratio	Cost	Ratio	Cost	Ratio
# Solvable	22		21		21	
SAMPLEAWARE cost	398.9	×0.28	606.4	×0.36	427.8	×0.32
ADAP cost	1422.3		1704.6		1328.7	
UNIFORM$_{C_{\text{ADAP}}}$ success rate	0.84		0.74		0.86	
DAP$_1$ cost	4187.0	×2.94	4591.0	×2.69	5246.0	×3.95
DAP$_2$ cost	2164.7	×1.52	2585.8	×1.52	2082.5	×1.57
DAP$_3$ cost	1792.7	×1.26	2260.1	×1.33	1819.5	×1.37
DAP$_5$ cost	1477.3	×1.04	1891.7	×1.11	1285.5	×0.97
DAP$_{10}$ cost	1169.4	×0.82	1463.2	×0.86	957.3	×0.72

B.3.2. COST-RATIO ROBUSTNESS

The main body fixes $c_{\text{ver}}/c_{\text{rew}} = 10$. We repeat the comparison for $c_{\text{ver}}/c_{\text{rew}} \in \{1, 10, 20, 30\}$ 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 grows as verification becomes more expensive: on coding, DAP$_1$ costs 3.6× more than ADAP at $c_{\text{ver}}/c_{\text{rew}} = 1$, rising to 7.2× at ratio 30. The UNIFORM$_{C_{\text{ADAP}}}$ rows confirm that a fixed policy given the same total budget as ADAP achieves only 79–90% success, underscoring that the savings come from genuine adaptivity rather than simply spending less. Notably, ADAP matches or outperforms DAP$_{10}$ on coding at all ratios and approaches it on math, without requiring any offline difficulty estimation or problem partitioning.

Table 3. Cost-ratio ablation on math (HMMT, 22 problems, 10 permutations each). The ratio column shows cost relative to ADAP at the same configuration.

	$c_{\text{ver}}/c_{\text{rew}} = 1$		$= 10$		$= 20$		$= 30$	
	Cost	Ratio	Cost	Ratio	Cost	Ratio	Cost	Ratio
SAMPLEAWARE cost	122.1	×0.50	398.9	×0.28	684.1	×0.26	965.4	×0.23
ADAP cost	245.2		1422.3		2627.4		4143.0	
UNIFORM$_{C_{\text{ADAP}}}$ success rate	0.80		0.84		0.84		0.85	
DAP$_1$ cost	866.0	×3.53	4187.0	×2.94	7877.0	×3.00	11567.0	×2.79
DAP$_2$ cost	530.0	×2.16	2164.7	×1.52	3868.8	×1.47	5572.9	×1.35
DAP$_3$ cost	480.9	×1.96	1792.7	×1.26	3098.6	×1.18	4404.5	×1.06
DAP$_5$ cost	420.9	×1.72	1477.3	×1.04	2528.6	×0.96	3578.2	×0.86
DAP$_{10}$ cost	357.0	×1.46	1169.4	×0.82	1945.5	×0.74	2718.2	×0.66

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.

	$c_{\text{ver}}/c_{\text{rew}} = 1$		$= 10$		$= 20$		$= 30$	
	Cost	Ratio	Cost	Ratio	Cost	Ratio	Cost	Ratio
SAMPLEAWARE cost	97.6	×0.40	179.9	×0.24	265.8	×0.25	349.6	×0.22
ADAP cost	241.4		747.6		1075.8		1564.1	
UNIFORM$_{C_{\text{ADAP}}}$ success rate	0.79		0.88		0.88		0.90	
DAP$_1$ cost	871.0	×3.61	4102.0	×5.49	7692.0	×7.15	11282.0	×7.21
DAP$_2$ cost	506.8	×2.10	2162.1	×2.89	3889.7	×3.62	5617.3	×3.59
DAP$_3$ cost	416.2	×1.72	1580.6	×2.11	2762.1	×2.57	3943.7	×2.52
DAP$_5$ cost	346.0	×1.43	1194.3	×1.60	2057.7	×1.91	2892.7	×1.85
DAP$_{10}$ cost	285.0	×1.18	787.3	×1.05	1276.1	×1.19	1742.9	×1.11

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 $\Pi_{\mathrm{str}}(\mathcal{D}, h^\star)$ 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 $V_i \mid R_i \sim \mathrm{Bernoulli}(h^\star(R_i))$. 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,

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

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

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

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

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

Therefore, verifying is optimal iff

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

Define

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

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 $c_{\mathrm{ver}} + (1 - h(R))J^*$). Thus J^* satisfies the Bellman identity

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

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

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

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

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

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,

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

If $h(r) \le \tau^\star$, 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 $S \le 1$ 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 $J^\star \ge c_{\text{ver}}/p$; a backward induction along the tail then gives $c_{\text{ver}}/p \le U \le J^\star$. Combined with $S \le 1$, this yields $0 \le S(pU - c_{\text{ver}}) \le pJ^\star - c_{\text{ver}}$, 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),

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

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,

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

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

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

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

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

Verify an above-threshold candidate. Among such candidates, verifying in decreasing order of h is optimal. Indeed, for $p \ge q$, 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 $c_{\text{ver}}(q - p) \le 0$. 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,

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

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 $0 \le W(P_t) \le J^\star$ gives $W(P) \le \mathbb{E}_\pi[C_T]$. 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.7, for every threshold t with $s_t > 0$,

$$\mathbb{E}[J(\mathcal{A}; h^\star, \mathcal{D})] \le 400 J_t.$$

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:

$$\mathbb{E}[J(\mathcal{A}; h^\star, \mathcal{D})] \le 400 \inf_{t:\, s_t > 0} J_t.$$

Using Lemma D.1 again,

$$\mathbb{E}[J(\mathcal{A}; h^\star, \mathcal{D})] \le 400 J^\star(h^\star, \mathcal{D}).$$

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 Lemma C.2, the optimal distribution-aware policy can be implemented as the streaming threshold policy from Lemma C.1. Let $\tau^\star$ be the corresponding break-even threshold. Thus the optimal policy verifies exactly those candidates whose score r satisfies

$$h^\star(r) > \tau^\star,$$

with ties at $h^\star(r) = \tau^\star$ being value-neutral. Since $h^\star$ is non-decreasing, the set

$$U^\star := \{r \in \mathbb{R} :\ h^\star(r) > \tau^\star\}$$

is an upper reward set.

Let

$$q_\star := \Pr_{R\sim\mathcal{D}}(R \in U^\star), \qquad s_\star := \mathbb{E}_{R\sim\mathcal{D}}\left[h^\star(R)\mathbf{1}\{R \in U^\star\}\right].$$

By Lemma C.1,

$$c_{\text{ver}}\mathbb{E}_{R\sim\mathcal{D}}\left[(h^\star(R) - \tau^\star)_+\right] = \tau^\star c_{\text{rew}}, \qquad J^\star(h^\star, \mathcal{D}) = \frac{c_{\text{ver}}}{\tau^\star}.$$

Since $U^\star = \{r : h^\star(r) > \tau^\star\}$,

$$\mathbb{E}\left[(h^\star(R) - \tau^\star)_+\right] = s_\star - \tau^\star q_\star.$$

Therefore

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

and hence

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

Thus the expected cost of the optimal upper-set policy is

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

It remains only to relate this upper-set policy to ordinary reward thresholds. Because $U^\star$ is an upper subset of $\mathbb{R}$, there exists a sequence of thresholds $(t_n)_{n\ge 1}$ such that

$$\{r:\ r \ge t_n\} \uparrow U^\star.$$

Let

$$q_n := q_{t_n}, \qquad s_n := s_{t_n}.$$

By monotone convergence,

$$q_n \to q_\star, \qquad s_n \to s_\star.$$

Moreover $s_\star > 0$, because otherwise $\mathbb{E}[(h^\star(R) - \tau^\star)_+] = 0$, contradicting $\tau^\star c_{\text{rew}} > 0$. Hence $s_n > 0$ for all sufficiently large n, and

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

Therefore

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

Conversely, for every threshold t with $s_t > 0$, the policy that repeatedly generates a candidate and verifies it iff $R \ge t$ has expected cost J_t. Since this is a valid distribution-aware policy,

$$J^\star(h^\star, \mathcal{D}) \le J_t \qquad \text{for every } t \text{ with } s_t > 0.$$

Taking the infimum over t gives

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

Combining the two inequalities proves the claim. □

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

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

Moreover $a_t \le b_t$, and $J_t \le B_{a_t,b_t} \le 4J_t$ 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 $s_t \le q_t$, we must have $a_t \le b_t$. For the first inequality, $s_t \ge 2^{-b_t}$ and $q_t \le 2^{-a_t}$ imply $J_t = \frac{c_{\text{rew}} + c_{\text{ver}} q_t}{s_t} \le (c_{\text{rew}} + c_{\text{ver}} 2^{-a_t}) 2^{b_t} = B_{a_t,b_t}$. 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 $B_{a_t,b_t} = (c_{\text{rew}} + c_{\text{ver}} 2^{-a_t}) 2^{b_t} < (c_{\text{rew}} + 2c_{\text{ver}} q_t) \frac{2}{s_t} \le 4J_t$. □

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 $k \le |A|$, and $k \le \ell \le |B|$. 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*

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

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

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

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

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

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

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

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 $u \ge 0$, $(a, b+u) \in S_{s_\star+u}$. Consequently, the stage in shell $s_\star + u$ has*

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

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 $(a,b) \in S_{s_\star}$, $2^{s_\star} c_{\min} \le B_{a,b} < 2^{s_\star+1} c_{\min}$. Multiplying by 2^u gives $2^{s_\star+u} c_{\min} \le B_{a,b+u} < 2^{s_\star+u+1} c_{\min}$. 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 $b^\star_{s_\star+u} \ge b+u$ and $j^\star_{s_\star+u} \ge b-a+u$. 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 $u \ge 0$, 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*

$$\rho_u = e^{-2^{u+1}} + e^{-2^u}.$$

Proof. By Lemma D.4, shell $s_\star + u$ generates at least $m_u := \lceil 2^{b+u+1} \rceil$ fresh samples and has verification budget at least $k_u := \lceil 6 \cdot 2^{b-a+u} \rceil$. 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

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

If $Z_t \ge 1$ and $N_t \le k_u$, 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 $Z_t \ge 1$ and $N_t \le k_u$ fails. Since $Z_t \sim \text{Bin}(m_u, s_t)$ and $s_t \ge 2^{-b}$, we have $m_u s_t \ge 2^{b+u+1} 2^{-b} = 2^{u+1}$. Thus

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

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

$$\mathbb{E}[N_t] \le (2^{b+u+1}+1)2^{-a} \le 3 \cdot 2^{b-a+u},$$

where the last inequality uses $b \ge a$. Let $\mu := 3 \cdot 2^{b-a+u}$. Since $k_u \ge 6 \cdot 2^{b-a+u} = 2\mu$, the Chernoff bound gives

$$\Pr(N_t > k_u) \le e^{-\mu/3} = e^{-2^{b-a+u}} \le e^{-2^u}.$$

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 $\Lambda_s \le 20 \cdot 2^s c_{\min}$.*

Proof. Using $\lceil x \rceil \le x+1$, we have

$$\Lambda_s \le c_{\text{rew}}(2^{b^\star_s+1}+1) + c_{\text{ver}}(6 \cdot 2^{j^\star_s}+1) \le 3c_{\text{rew}} 2^{b^\star_s} + 7c_{\text{ver}} 2^{j^\star_s}.$$

Since $b^\star_s$ is attained by some pair in S_s, $c_{\text{rew}} 2^{b^\star_s} < 2^{s+1} c_{\min}$. Similarly, since $j^\star_s$ is attained by some pair in S_s, $c_{\text{ver}} 2^{j^\star_s} < 2^{s+1} c_{\min}$. Therefore $\Lambda_s < 3 \cdot 2^{s+1} c_{\min} + 7 \cdot 2^{s+1} c_{\min} = 20 \cdot 2^s c_{\min}$. □

Lemma D.7. *For every threshold t with $s_t > 0$, the expected cost of the shellwise algorithm is at most $400J_t$.*

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, $2^{s_\star} c_{\min} \le B_{a,b} \le 4J_t$.

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

$$\sum_{s=0}^{s_\star-1} \Lambda_s \le 20 c_{\min} \sum_{s=0}^{s_\star-1} 2^s < 20 \cdot 2^{s_\star} c_{\min}.$$

Now consider the cost from shell $s_\star$ onward. The algorithm reaches shell $s_\star$ with probability at most one. For $u \ge 1$, 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

$$\mathbb{E}[\text{cost from shell } s_\star \text{ onward}] \le 20 \cdot 2^{s_\star} c_{\min} \left(1 + \sum_{u \ge 1} 2^u \rho_{u-1}\right).$$

Since $\rho_{u-1} = e^{-2^u} + e^{-2^{u-1}}$, Lemma D.8 gives

$$\mathbb{E}[\text{cost from shell } s_\star \text{ onward}] \le 80 \cdot 2^{s_\star} c_{\min}.$$

Combining the two parts,

$$\mathbb{E}[J(\mathcal{A}; h, D)] \le 100 \cdot 2^{s_\star} c_{\min} \le 100 B_{a,b} \le 400 J_t.$$

□

Lemma D.8. *Let $\Gamma = 1 + \sum_{u \ge 1} 2^u (e^{-2^u} + e^{-2^{u-1}})$. Then $\Gamma \le 4$.*

Proof. We use $\sum_{n \ge 1} n e^{-n} = e^{-1}/(1 - e^{-1})^2 < 1$. Since $\{2^u : u \ge 1\} \subseteq \mathbb{N}$, $\sum_{u \ge 1} 2^u e^{-2^u} \le \sum_{n \ge 1} n e^{-n} < 1$. Also, with $n_u = 2^{u-1}$, we have $2^u e^{-2^{u-1}} = 2 n_u e^{-n_u}$, and hence $\sum_{u \ge 1} 2^u e^{-2^{u-1}} \le 2 \sum_{n \ge 1} n e^{-n} < 2$. Thus $\Gamma < 1 + 1 + 2 = 4$. □

E. Appendix of Section 6

Proof of Theorem 6.2. Fix an integer $m \le \mathfrak{s}_0$. 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 $\mathcal{D}(x_i) = 1/m$,

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

Algorithm lower bound. Place a uniform prior $I \sim \text{Unif}([m])$ 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

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

Multiplying the survival probabilities,

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

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

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

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

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

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 $D \in \Delta(\mathcal{X})$,

$$\Pr\left(\operatorname{err}_D(\widehat{h}, h^\star) \leq \varepsilon\right) \geq 1 - \delta,$$

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 $\Pr_{X\sim D}(h^\star(X)=1) > 0$,

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

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

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

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

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

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

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

the triangle inequality implies that $T=1$ forces

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

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

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

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

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

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 $\mathcal{A}_{\mathrm{search}}$ 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 $\Omega(d)$.

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

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

if $\mathfrak{s}_0 > \frac{c_{\text{ver}}}{c_{\text{rew}}}$ **then**
  while true **do**
    Generate $Z \sim \mathcal{D}$, pay c_{rew}, verify Z, and pay c_{ver}
    if the verifier returns 1 **then**
      return Z
else
  $n \leftarrow \lceil \frac{c_{\text{ver}}}{c_{\text{rew}}} \mathfrak{s}_0 \rceil$
  while true **do**
    Generate $(Z_1, \ldots, Z_n) \sim \mathcal{D}^{\otimes n}$ and pay $n c_{\text{rew}}$
    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

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 $B_1, \ldots, B_d \overset{i.i.d.}{\sim} \mathrm{Bernoulli}(\frac{1}{2})$. 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. Algorithm and Proof of Theorem 6.3

Proof of Theorem 6.3. Soundness is immediate since $\mathcal{A}_{\mathrm{CS}}$ 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 $\mathcal{A}_{\mathrm{CS}}$ enters the verify-all branch, i.e. $\mathfrak{s}_0 > c_{\mathrm{ver}}/c_{\mathrm{rew}}$. The branch succeeds with geometric success probability p, so

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

Therefore

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

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

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

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 Equation (8), we have $\mathbb{E}[J(\mathcal{A}_{\mathrm{CS}}; h^\star, \mathcal{D})] \le \frac{2B}{np} = \frac{2(B/n)}{p} \le \frac{4c_{\mathrm{rew}}}{p}$. Then, notice that $\frac{4c_{\mathrm{rew}}}{p} \le 4J^\star(h^\star, \mathcal{D}) \le 6\mathfrak{s}_0 J^\star(h^\star, \mathcal{D})$, 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 $T_+ := \inf\{i \ge 1 : h^\star(Z_i) = 1\}$, where $Z_i i.i.d. \mathcal{D}$, then $\mathbb{E}[T_+] = 1/p$, and hence every sound policy has expected cost at least $c_{\text{rew}}/p + c_{\text{ver}}$. □