NOT ALL THOUGHTS ARE WORTH THINKING: RELIABILITY-AWARE TEST-TIME SCALING

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time scaling improves LLM reasoning, but concept-guided inference can waste compute on weak or redundant solving directions. We study *pre-expansion concept allocation*: which explored concepts should receive solution-generation compute? We introduce RATTS, a **Reliability-Aware Test-Time Scaling** method that combines calibrated predictions of judged concept usefulness with semantic soft coverage. Its objective favors useful, complementary concepts, is monotone submodular, and admits a $(1 - 1/e)$ greedy approximation guarantee; under an explicit concept-success model, we further derive a conditional bound connecting the surrogate to candidate recall. Across MATH500, GPQA DIAMOND, and HUMANEVAL+ with three generators, RATTS achieves 95.65% macro-average pass-any at $K_{\max} = 50$ and $\rho = 0.30$, versus 85.73% for score-only selection and 94.45% for uniform-weight semantic coverage. RATTS retains 98.7% of full-expansion pass-any while reducing exploration-plus-expansion tokens by 56.69%. Macro-average final accuracy is 74.42% versus 74.08% for full expansion, with a paired difference of 0.34 pp (95% CI: $[-0.90, 1.62]$). To enable reproduction of these results, we will release the code and experimental artifacts upon acceptance.

1 INTRODUCTION

Test-time scaling improves LLM reasoning by spending additional inference compute, often through sampling and aggregation of multiple solutions (Wang et al., 2023; Brown et al., 2024). Concept-guided inference makes this process more structured: it first explores high-level solving directions and then expands them into complete solutions (Handa et al., 2026). Yet exploration alone does not determine where subsequent compute should go. Some concepts are promising, others are weak, and several may encode essentially the same strategy. Expanding all of them wastes compute, while ranking them independently can select redundant directions. We therefore study *pre-expansion concept allocation*: given an explored concept pool and a fixed budget, which reasoning directions should receive solution-generation compute before their outcomes are known?

This decision is relevant whenever inference is constrained by tokens, latency, serving cost, or compute. A deployed reasoning system may generate several plausible approaches but be unable to fully expand all of them under a fixed response-time or resource budget. Selective expansion can improve the efficiency of reasoning pipelines without changing the underlying generator: compute is concentrated on a smaller set of promising, complementary directions. Beyond reducing generation cost, this type of inference-time allocation can help make stronger reasoning systems more practical under fixed deployment budgets, from interactive assistants to scientific and coding systems. RATTS does not assume that more compute is always available; instead, it asks how to use a given inference budget more effectively.

We introduce RATTS, a *Reliability-Aware Test-Time Scaling* method for budgeted concept selection (Figure 1). **Reliability-aware concept allocation** refers to allocating expansion budget using calibrated estimates of concept usefulness, so that selection accounts not only for semantic diversity but also for the calibrated confidence that a concept is useful enough to warrant expansion. A lightweight scorer provides these calibrated usefulness estimates, while semantic similarity captures overlap between reasoning directions. RATTS combines the two signals through a relevance-weighted soft-coverage objective: concepts receive high value when they are predicted to be useful

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

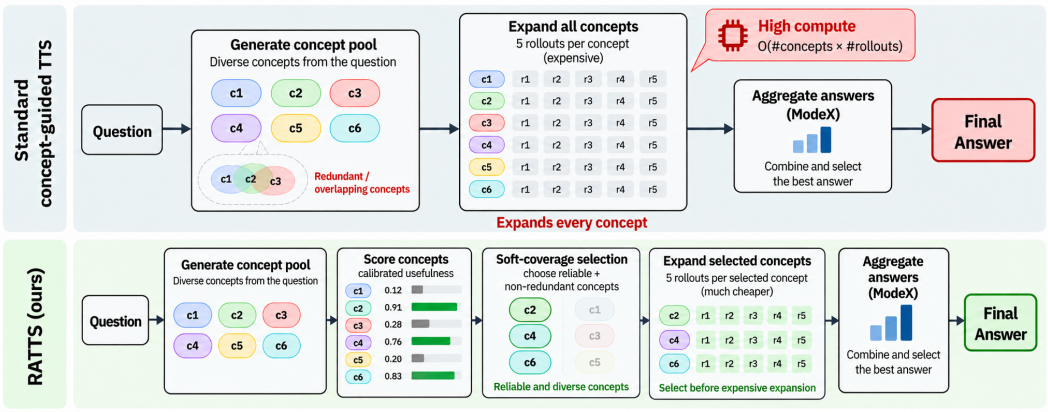


Figure 1: **Reliability-aware concept allocation with RATTS.** RATTS selects useful, complementary concepts before expensive expansion. At $K_{\max} = 50$ and $\rho = 0.30$, it retains 98.7% of full-expansion pass-any while reducing expansion tokens by 69.23% and total exploration-plus-expansion tokens by 56.69% across nine settings.

and cover semantic mass not already represented by the selected set. The expansion budget is therefore allocated to directions that are jointly promising and complementary.

This objective is designed to reflect the practical goal of preserving useful candidate solutions while avoiding redundant expansion. Candidate recall itself is monotone submodular under arbitrary dependence between concept-success events, while the RATTS surrogate is also monotone submodular and admits the standard $(1 - 1/e)$ greedy guarantee. Under an explicit concept-success model, retained candidate recall has the same product-form structure as the surrogate, yielding a conditional bound in terms of relevance- and coverage-estimation error. We next ask whether this allocation strategy preserves useful candidates while reducing inference cost in practice. Across 3 benchmarks and 3 generators, RATTS achieves 95.65% macro-average pass-any at $K_{\max} = 50$ and $\rho = 0.30$, while substantially reducing generation compute relative to full expansion. Macro-average final accuracy remains comparable to full expansion (74.42% vs. 74.08%).

Verifier-based approaches score complete solutions or intermediate reasoning steps to improve selection or search (Cobbe et al., 2021; Lightman et al., 2024), while adaptive test-time scaling methods decide *how much* inference compute to spend based on budget, problem difficulty, response agreement, or confidence (Snell et al., 2025; Wu et al., 2025; Aggarwal et al., 2023; Kim et al., 2026a). RATTS instead addresses an outcome-free allocation decision: given a fixed expansion budget and already explored reasoning directions, *where* should that compute be spent before any solution rollout is observed?

Structured reasoning methods explore multiple solution paths (Yao et al., 2023) or suppress redundant branches using semantic similarity (Kim et al., 2025). Most closely related, GuidedSampling (Handa et al., 2026) separates concept exploration from solution generation; RATTS takes this decomposition as given and formalizes the allocation decision between the two stages. Its selection rule also differs from classical relevance–diversity methods such as MMR (Carbonell & Goldstein, 1998), which trade relevance against similarity to previously selected items. RATTS instead optimizes residual relevance-weighted coverage of the full concept pool, linking this pre-expansion surrogate to retained candidate recall under the model in Section 3.2. See Appendix A for broader discussion.

Contributions. **(i)** We identify and formalize *pre-expansion concept allocation* as a distinct test-time scaling problem: deciding which explored reasoning directions should receive scarce solution-generation compute. **(ii)** We introduce RATTS, which combines calibrated judged-usefulness with semantic coverage to select useful, complementary concepts. We show that candidate recall itself is monotone submodular under arbitrary dependence between concept-success events, and derive a product-form pre-expansion surrogate with a conditional connection to retained candidate recall. **(iii)** Across nine dataset–generator settings, we show that RATTS substantially improves over independent score-based allocation while preserving nearly all full-expansion candidate recall at substantially lower generation cost and comparable observed final-answer accuracy.

2 METHODOLOGY

We formulate concept-guided test-time scaling (Handa et al., 2026) as a *pre-expansion allocation* problem over candidate reasoning concepts. RATTS operates after concept exploration and before solution generation, combining judged-usefulness estimates with semantic coverage to decide which concepts receive expansion compute. The allocation is outcome-free: no solution rollout or correctness signal is observed before the expansion set is chosen. The concept generator, solution generator, and downstream aggregation rule remain fixed; only the allocation of solution-generation compute changes.

2.1 CONCEPT EXPANSION AS BUDGETED SELECTION

For a problem x, let $C_x = (c_1, \ldots, c_{K_x})$ denote the available concept pool, where each c_i specifies a candidate solving direction. Expanding a subset $S \subseteq [K_x]$ with M rollouts per concept produces the indexed candidate collection

$$\mathcal{Y}_x(S) = \big((c_i, y_{i,m})\big)_{i \in S,\, m \in [M]}, \qquad y_{i,m} \sim p_\theta(\cdot \mid x, c_i),$$

where $[n] = \{1, \ldots, n\}$. Full expansion uses $S = [K_x]$ and requires $K_x M$ rollouts.

Given a concept budget $B_x \leq K_x$, a selection rule chooses $S_x \subseteq [K_x]$ with $|S_x| \leq B_x$ before generating solution rollouts. We parameterize this budget as $B_x = \lceil \rho K_x \rceil$, where $\rho \in (0, 1]$ is the expansion fraction. Only the selected concepts are expanded, requiring $M|S_x|$ rollouts. Defining

$$P_x(S) := \Pr(\mathcal{Y}_x(S) \text{ contains a correct solution}),$$

the ideal pre-expansion allocation solves

$$S_x^\star \in \arg\max_{\substack{S \subseteq [K_x] \\ |S| \leq B_x}} P_x(S).$$

Because $P_x(S)$ depends on unobserved rollout outcomes, RATTS instead constructs a surrogate using only information available before expansion. A fixed aggregation rule δ returns the final prediction,

$$\hat{y}_{\mathrm{RATTS}}(x) = \delta(x, \mathcal{Y}_x(S_x)).$$

We use ModeX (Choi & Li, 2026) for δ in our evaluation and report candidate recall (pass-any) and final-answer accuracy separately. The former measures whether the candidate collection contains a correct solution, while the latter measures whether the returned answer is correct.

2.2 CALIBRATED CONCEPT RELEVANCE

RATTS estimates concept usefulness from supervised problem–concept pairs. Here, reliability refers to calibrated prediction of judged concept usefulness, rather than calibrated probability of rollout success. Each training example (x, c, z) has an ordinal label $z = z(x, c) \in \mathcal{Z}$, where $\mathcal{Z} = \{0, 0.25, 0.5, 0.75, 1.0\}$ and larger values indicate more useful solving directions. Labels are obtained from an LLM judge, providing supervision without requiring the generation and verification of full solution rollouts for every training concept.

The reliability model maps (x, c) to logits $h_\phi(x, c) \in \mathbb{R}^{|\mathcal{Z}|}$, defining a predictive distribution and an expected usefulness score:

$$\widehat{p}_\phi(z \mid x, c) = \mathrm{softmax}(h_\phi(x, c))_z, \qquad \widehat{s}_\phi(x, c) = \sum_{z \in \mathcal{Z}} z\, \widehat{p}_\phi(z \mid x, c).$$

To learn usefulness levels and encourage their within-problem ordering, we use the refinement objective

$$\mathcal{L}_{\mathrm{ref}}(\phi) = \mathcal{L}_{\mathrm{cls}}(\phi) + \lambda_{\mathrm{score}} \mathcal{L}_{\mathrm{score}}(\phi) + \lambda_{\mathrm{rank}} \mathcal{L}_{\mathrm{rank}}(\phi).$$

For the training data $\mathcal{D}_{\mathrm{tr}}$, its components are

$$\mathcal{L}_{\mathrm{cls}}(\phi) = -\sum_{(x,c,z) \in \mathcal{D}_{\mathrm{tr}}} w_z \log \widehat{p}_\phi(z \mid x, c), \quad \mathcal{L}_{\mathrm{score}}(\phi) = \sum_{(x,c,z) \in \mathcal{D}_{\mathrm{tr}}} \ell_{\mathrm{reg}}(\widehat{s}_\phi(x, c) - z),$$

$$\mathcal{L}_{\mathrm{rank}}(\phi) = \sum_x \sum_{\substack{(c_a, c_b): \\ z(x,c_a) > z(x,c_b)}} \log\big(1 + \exp\big[-\big(\widehat{s}_\phi(x, c_a) - \widehat{s}_\phi(x, c_b)\big)\big]\big).$$

Here w_z is a class weight, ℓ_{reg} is a smooth regression loss, and the ranking term compares labeled concepts from the same training problem. Classification fits the discrete usefulness labels, while regression and ranking encourage agreement with their numerical values and relative ordering.

We apply post-hoc vector scaling to improve calibration against held-out judged-usefulness labels. With ϕ fixed, we fit

$$\widetilde{p}_{\phi,\eta}(z \mid x, c) = \text{softmax}(a \odot h_\phi(x, c) + b)_z, \qquad \eta = (a, b),$$

where $a, b \in \mathbb{R}^{|\mathcal{Z}|}$, by minimizing negative log-likelihood on a held-out development split. The scorer and calibrator are fixed before benchmark evaluation.

The calibrated relevance score is

$$p_{\text{rel}}(x, c) = \sum_{z \in \{0.75, 1.0\}} \widetilde{p}_{\phi,\eta}(z \mid x, c).$$

Combining the two highest usefulness levels includes both directly useful and useful-but-incomplete solving directions. Calibration targets judged usefulness rather than rollout success, and p_{rel} serves as an inexpensive proxy for expansion utility. Because the scores are subsequently normalized within each problem, selection is invariant to a common rescaling of all p_{rel} values; vector scaling affects selection only through changes in their relative magnitudes or ordering.

2.3 Soft-Coverage Selection

Independent ranking by relevance does not account for overlap between solving directions. RATTS therefore combines the relevance scores with semantic coverage, selecting directly from the available concept pool without an additional hard-clustering stage.

For a fixed problem x, write $r_i = p_{\text{rel}}(x, c_i)$ and normalize the scores as

$$\mu_i = \frac{r_i}{\sum_{j=1}^{K_x} r_j},$$

using $\mu_i = 1/K_x$ when the denominator is zero. These weights determine how much coverage mass is assigned to each concept.

Let e_i be the L2-normalized embedding of c_i, obtained by attention-mask-weighted mean pooling of the token representations from the frozen `BAAI/bge-small-en-v1.5` encoder (Xiao et al., 2023). We define

$$\kappa_{ij} = \text{clip}\big(e_i^\top e_j, 0, 1\big), \qquad \kappa_{ii} = 1.$$

For $S \subseteq [K_x]$, the relevance-weighted soft-coverage objective is

$$\widehat{F}_x(S) = \sum_{i=1}^{K_x} \mu_i \left(1 - \prod_{j \in S} (1 - \kappa_{ij})\right). \tag{1}$$

For each concept i, the product $\prod_{j \in S}(1 - \kappa_{ij})$ represents its residual uncovered mass, so $1 - \prod_{j \in S}(1 - \kappa_{ij})$ acts as a soft noisy-OR coverage quantity. The objective therefore rewards coverage of high-relevance regions while naturally producing diminishing returns for redundant selections. Section 3 shows that, under an explicit concept-success model, retained candidate recall has the same product-sum structure.

RATTS maximizes this objective approximately using greedy selection under the budget B_x. For an unselected concept j, the marginal gain is

$$\Delta_x(j \mid S) = \widehat{F}_x(S \cup \{j\}) - \widehat{F}_x(S) = \sum_{i=1}^{K_x} \mu_i \, \kappa_{ij} \prod_{h \in S} (1 - \kappa_{ih}).$$

Starting from $S^{(0)} = \emptyset$, for $t = 0, \ldots, B_x - 1$, we select and update

$$j_t \in \arg\max_{j \in [K_x] \setminus S^{(t)}} \Delta_x(j \mid S^{(t)}), \qquad S^{(t+1)} = S^{(t)} \cup \{j_t\}.$$

The resulting set $\hat{S}_x = S^{(B_x)}$ is expanded with M rollouts per concept. Relevance determines where coverage mass is placed, while semantic overlap reduces the marginal value of redundant selections. Maintaining the residual vector

$$u_i(S) = \prod_{h \in S} (1 - \kappa_{ih})$$

allows all marginal gains to be evaluated in $O(K_x^2)$ time per greedy step, for $O(B_x K_x^2)$ selection time after constructing the similarity matrix and $O(K_x^2)$ space.

3 Theoretical Analysis

We relate the practical soft-coverage objective to the candidate-recall quantity that concept selection seeks to preserve. We first show that candidate recall itself is monotone submodular under arbitrary dependence between concept-success events. We then introduce a success-mode model under which retained candidate recall has the same product-sum form as the RATTS surrogate, allowing their discrepancy to be quantified. Our guarantees concern pass-any rather than final-answer aggregation; proofs are in Appendix B.

Fix a problem x, its concept pool C_x, and M rollouts per concept. Define all potential rollouts on a common probability space. Writing $R(x, y)$ for correctness, let

$$A_i(x) = \{\exists m \le M : R(x, y_{i,m}) = 1\}, \qquad H_x = \bigcup_{i=1}^{K_x} A_i(x).$$

For $S \subseteq [K_x]$, define

$$P_x(S) = \mathbb{P}\left(\bigcup_{i \in S} A_i(x)\right), \qquad \mathrm{Ret}_x(S) = \frac{P_x(S)}{P_x([K_x])} = \mathbb{P}\left(\bigcup_{i \in S} A_i(x) \,\middle|\, H_x\right),$$

where Ret_x is defined when $P_x([K_x]) > 0$.

Proposition 3.1 (Candidate-recall submodularity)**.** *Under any fixed joint rollout law, P_x is normalized, monotone, and submodular. If $P_x([K_x]) > 0$, then Ret_x has the same properties.*

No independence assumption is required. Since P_x depends on success probabilities unavailable before expansion, RATTS instead optimizes the surrogate in Eq. (1).

3.1 Greedy Soft-Coverage Selection

Proposition 3.2 (Greedy soft-coverage selection)**.** *For fixed weights $\mu_i \ge 0$, $\sum_i \mu_i = 1$, and similarities $\kappa_{ij} \in [0, 1]$, the objective $\widehat{F}_x$ is normalized, monotone, and submodular. For an integer budget $1 \le B_x \le K_x$, let S_x^{cov} maximize $\widehat{F}_x(S)$ over $|S| \le B_x$. Greedy selection returns $\hat{S}_x$ satisfying*

$$\widehat{F}_x(\hat{S}_x) \ge \gamma_{B_x} \widehat{F}_x(S_x^{\mathrm{cov}}), \qquad \gamma_B = 1 - \left(1 - \frac{1}{B}\right)^B \ge 1 - \frac{1}{e}.$$

Let

$$\Delta_x^{(1)}(\hat{S}_x) \ge \Delta_x^{(2)}(\hat{S}_x) \ge \cdots$$

be the remaining marginal gains, and let

$$m_x = \min\{B_x, K_x - |\hat{S}_x|\}.$$

Then

$$\widehat{F}_x(S_x^{\mathrm{cov}}) - \widehat{F}_x(\hat{S}_x) \le \sum_{k=1}^{m_x} \Delta_x^{(k)}(\hat{S}_x) \le B_x \Delta_x^{\max}(\hat{S}_x),$$

where

$$\Delta_x^{\max}(\hat{S}_x) = \max_{j \notin \hat{S}_x} \left[\widehat{F}_x(\hat{S}_x \cup \{j\}) - \widehat{F}_x(\hat{S}_x)\right],$$

with value zero when $\hat{S}_x = [K_x]$.

The first bound is the standard greedy guarantee; the second gives an a posteriori certificate for the implemented surrogate.

3.2 Conditional Guarantees for Candidate Recall

We now connect the surrogate to candidate recall. Under the following model, retained pass-any has exactly the same product-sum form as the RATTS objective.

Success-mode assumption. Conditional on H_x, suppose there is a latent mode $Z_x \in [K_x]$ with

$$\pi_i = \mathbb{P}(Z_x = i \mid H_x), \qquad \pi_i \geq 0, \qquad \sum_i \pi_i = 1.$$

Let

$$\mathcal{I}_x = \{i \in [K_x] : \pi_i > 0\}.$$

For each $i \in \mathcal{I}_x$, suppose $A_1(x), \ldots, A_{K_x}(x)$ are conditionally independent given $H_x, Z_x = i$, with

$$q_{ij} = \mathbb{P}(A_j(x) \mid H_x, Z_x = i), \qquad q_{ij} \in [0,1], \qquad q_{ii} = 1.$$

Then

$$\mathrm{Ret}_x(S) = \sum_{i \in \mathcal{I}_x} \pi_i \left[1 - \prod_{j \in S}(1 - q_{ij})\right]. \tag{2}$$

This has the same product-sum form as Eq. (1) and coincides with it when $\mu_i = \pi_i$ and $\kappa_{ij} = q_{ij}$.

For approximate quantities, define

$$d_{\mu,x} = \frac{1}{2}\sum_{i=1}^{K_x} |\mu_i - \pi_i|, \qquad d_{\kappa,x} = \max_{\substack{i \in \mathcal{I}_x \\ j \in [K_x]}} |\kappa_{ij} - q_{ij}|,$$

and

$$\eta_x(B_x) = d_{\mu,x} + B_x d_{\kappa,x}.$$

The q_{ij} need not be symmetric, whereas $\kappa_{ij} = \kappa_{ji}$; hence $d_{\kappa,x}$ also captures mismatch between undirected semantic similarity and directed success transfer.

Theorem 3.3 (Conditional retained-pass-any guarantee)**.** *Under the success-mode assumption, fix $1 \leq B_x \leq K_x$, and let $S_x^\star$ maximize $\mathrm{Ret}_x(S)$ over $|S| \leq B_x$. Then*

$$\sup_{|S| \leq B_x} \left|\mathrm{Ret}_x(S) - \widehat{F}_x(S)\right| \leq \eta_x(B_x),$$

and the greedy RATTS set satisfies

$$\mathrm{Ret}_x(\hat{S}_x) \geq \gamma_{B_x}\mathrm{Ret}_x(S_x^\star) - (1 + \gamma_{B_x})\eta_x(B_x). \tag{3}$$

Moreover,

$$1 - \mathrm{Ret}_x(\hat{S}_x) \leq 1 - \widehat{F}_x(\hat{S}_x) + \eta_x(B_x). \tag{4}$$

When $\eta_x(B_x) = 0$, greedy RATTS achieves a γ_{B_x} fraction of the optimal retained pass-any at the same budget. For approximate parameters, Eq. (3) separates weight and coverage mismatch, while Eq. (4) relates retained recall to uncovered surrogate mass.

Relation to the practical scorer. The implementation obtains μ from calibrated judged-usefulness predictions and κ from semantic embeddings. These quantities need not satisfy the success-mode assumption or yield small $d_{\mu,x}$ and $d_{\kappa,x}$. Thus, Theorem 3.3 gives a conditional structural connection rather than a numerical certificate. Proposition 3.1 applies directly to candidate recall, while Proposition 3.2 applies directly to the implemented surrogate. Candidate retention is evaluated empirically.

4 Experiments

We evaluate whether RATTS preserves candidate recall while reducing solution-expansion compute. Concept generation is held fixed, and selectors operate on the same explored pools and stored solution samples, isolating the pre-expansion allocation decision. We evaluate across concept-pool sizes and rollout budgets, measuring candidate recall, final-answer accuracy, and computational cost. Section 5 presents the main results, with detailed analyses deferred to the appendix.

4.1 Benchmarks and Generators

We evaluate on `Math500` (500 problems), `GPQA Diamond` (198 problems), and `HumanEval+` (164 problems), covering mathematical reasoning, scientific question answering, and programming, respectively. We evaluate each benchmark with `Gemma4-E4B`, `Qwen3.5-9B`, and `GLM-4.7-Flash`, yielding nine dataset–generator settings. We set the maximum available concept-pool target to $K_{\max} = 50$ and generate $M = 5$ solution rollouts for each expanded concept.

4.2 Reliability-Model Training

For each generator, we train a reliability model with a BGE-small backbone on concepts generated from the `s1K` training problems (Muennighoff et al., 2025). An LLM judge assigns the ordinal usefulness labels described in Section 2.2, allowing us to construct the training data without generating and verifying full solution rollouts for every training concept. After training, we calibrate the model's predictions using vector scaling on a held-out development split.

Each generator-specific scorer and its calibrator are fixed before benchmark evaluation and applied to all three benchmarks without benchmark-specific training or recalibration. `HumanEval+` therefore evaluates transfer of the `s1K`-trained scorer to programming. Benchmark examples and their correctness labels are not used to train or calibrate the scorer. The appendix describes the supervision procedure, judge prompts, data splits, and generator-specific training configurations.

4.3 Baselines and Ablations

For concept-count comparisons, we evaluate full Guided Sampling (Handa et al., 2026), Best Score Concepts, Top Cluster Representatives, Random Cluster, SkyWork PRM, and ReasonFlux PRM. Best Score Concepts uses the same frozen scorer, calibrated relevance scores p_{rel}, concept pool, and budget $B_x = \lceil \rho K_x \rceil$ as RATTS, but selects the top-B_x concepts independently, providing a score-only control without semantic coverage.

At the main operating point, we additionally compare against uniform-weight soft coverage and Maximal Marginal Relevance (MMR). Uniform-weight coverage sets $\mu_i = 1/K_x$ while retaining the same semantic similarities and budget, isolating the contribution of learned relevance weighting within the soft-coverage selector. MMR uses the same calibrated relevance scores and semantic similarities and provides a classical relevance–diversity alternative; we use $\lambda = 0.50$ for comparison.

SkyWork PRM and ReasonFlux PRM are adapted as pre-expansion concept scorers and select concepts by independent PRM ranking. These comparisons therefore evaluate alternative concept-scoring adaptations rather than coverage-augmented PRM variants.

For rollout-budget comparisons, we additionally evaluate ASC, MMC, ReASC, and SSDP. We further ablate refinement, calibration, and both components. Random Cluster results are averaged over three seeds. The appendix provides full baseline configurations, budget settings, and ablation details.

4.4 Evaluation Protocol

We use offline replay to evaluate all methods on identical concept pools and solution samples. For each problem, we store the valid concepts and five rollouts per concept. RATTS selects concepts using only pre-expansion information, after which replay retains the corresponding stored rollouts.

We evaluate two budget regimes. In the *concept-count sweep*, we vary the available prefix target $K \leq K_{\max}$ with $\rho = 0.30$. In the *relative-budget sweep*, we fix $K_{\max} = 50$ and vary ρ. If K_x valid concepts are available for problem x, RATTS expands $B_x = \lceil \rho K_x \rceil$, while full Guided Sampling expands all K_x concepts.

Rollout-budget curves report mean per-problem consumption, $N_a(x)/(MK_x)$, accounting for variable pool sizes and budget rounding. For final-answer accuracy, ModeX is held fixed across selectors. Reference answers are used only for evaluation: correctness is measured by parsed-answer matching on `Math500` and `GPQA Diamond`, and execution-based testing on `HumanEval+`. Additional generation, filtering, replay, and aggregation details are provided in the appendix.

4.5 Metrics and Computational Cost

Our primary allocation metric is pass-any, which measures whether the selected expansion set retains at least one correct candidate. We additionally report final-answer accuracy, token reduction, and selector latency. Macro averages weight the nine dataset–generator settings equally. Detailed metric definitions, token accounting, latency measurement, and the complete statistical protocol (including paired bootstrap intervals, McNemar tests, and Holm correction) are provided in the supplemental material; see Appendix C.10.

5 Results

Figure 2 shows that RATTS benefits from larger concept pools while expanding only a fraction of the available concepts. At $K_{\max} = 50$ and $\rho = 0.30$, its macro-average pass-any is 95.65%. Independent score-only selection using the same relevance scores and concept budget achieves 85.73%, whereas uniform-weight soft coverage reaches 94.45%. Thus, semantic coverage alone performs substantially better than independent ranking in this evaluation, while learned usefulness weighting provides a further 1.20-percentage-point improvement within the same soft-coverage selector (95% CI: $[0.91, 1.52]$). Random Cluster achieves 94.61%, further illustrating the value of avoiding redundant expansion.

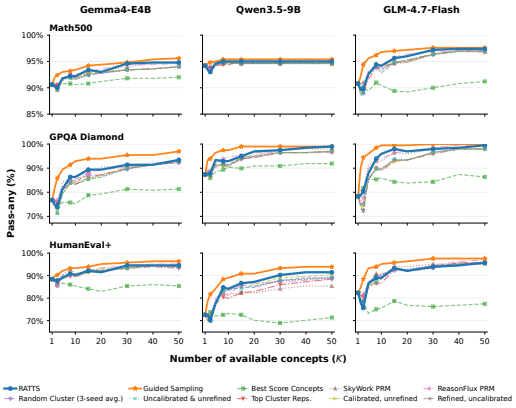


Figure 2: Pass-any as the available concept pool grows. RATTS expands $B_x = \lceil 0.30 K_x \rceil$ concepts, whereas Guided Sampling expands all K_x concepts. An enlarged version is provided as Figure 4 in the appendix.

Coverage and relevance weighting. A standard Maximal Marginal Relevance (MMR) selector using the same relevance scores, semantic similarities, and concept budget achieves 94.55% macro-average pass-any at $\lambda = 0.50$, compared with 95.65% for RATTS. This provides a matched classical relevance–diversity comparison using the same underlying relevance and similarity signals. On `HumanEval+` with `Qwen3.5-9B`, RATTS achieves 91.46%, versus 71.34% for independent score-only selection, despite using a scorer trained on mathematical concept judgments.

RATTS also has higher observed macro-average pass-any than the evaluated PRM-based concept selectors: 95.65%, compared with 94.75% for SkyWork and 95.12% for ReasonFlux. The difference from ReasonFlux is 0.53 percentage points with a paired 95% CI of $[-0.08, 1.15]$, and is therefore not statistically resolved. RATTS uses a substantially smaller 33M-parameter encoder than the evaluated 1.5B PRMs and has lower measured mean complete-selector latency: 333.058 ms per problem, compared with 626.940 ms for SkyWork and $1{,}886.066$ ms for ReasonFlux. For SkyWork, the difference is primarily in the latency tail rather than the median; the appendix reports the complete timing breakdown and execution configurations.

High candidate recall with reduced expansion. Figure 3 shows that RATTS maintains high pass-any at low rollout consumption, particularly on `GPQA Diamond` and `HumanEval+`. Over the shared budget ranges used for each paired comparison, RATTS achieves higher normalized pass-any AUC than ASC, MMC, ReASC, and the offline SSDP replay adaptation by 3.13–5.87 percentage points, with paired 95% confidence intervals entirely above zero.

The corresponding final-accuracy AUC comparison is different. RATTS is higher than the offline SSDP replay adaptation, whereas the differences from ASC, MMC, and ReASC are not statistically resolved. This distinction reflects the different quantities being measured: pass-any asks whether the retained pool contains a correct candidate, while consensus-based stopping rules and downstream

ModeX depend on the distribution and agreement structure of retained responses. Higher candidate recall therefore need not translate into higher consensus accuracy.

At the main operating point, RATTS achieves 69.23% lower expansion-token cost and 56.69% lower exploration-plus-expansion token cost under the recorded replay accounting. Alongside these savings, RATTS achieves 74.42% macro-average final-answer accuracy, compared with 74.08% for full Guided Sampling. The observed difference is 0.34 percentage points, with a paired 95% bootstrap confidence interval of $[-0.90, 1.62]$. The interval includes zero, so the average accuracy difference is not statistically resolved. Final accuracy exceeds full expansion in seven of the nine settings, matches it in one, and is lower in one.

The exception is `HumanEval+` with `GLM-4.7-Flash`, where final accuracy decreases from 77.44% under full expansion to 71.95% with RATTS. This is an out-of-domain transfer setting for the relevance scorer, which is trained on mathematical concept judgments. Selective expansion also changes the candidate composition seen by ModeX, so retaining a correct candidate does not guarantee that the same consensus answer will be selected. The present evaluation does not isolate which effect accounts for this setting-specific decrease.

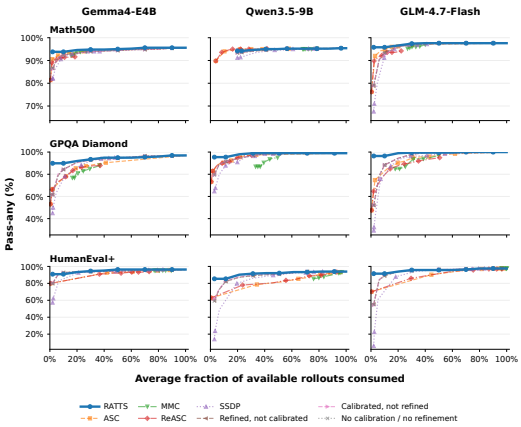


Figure 3: Pass-any versus the mean per-problem fraction of available rollouts consumed at $K_{\max} = 50$. RATTS varies its expansion fraction ρ, while comparison methods vary their budget or stopping parameters. An enlarged version is provided as Figure 6 in the appendix.

Overall, RATTS retains 98.7% of full-expansion pass-any while substantially reducing generation cost. The progression from 85.73% with independent relevance ranking to 94.45% with uniform soft coverage and 95.65% with full RATTS shows that uniform coverage recovers most of the observed gap to RATTS, while learned judged-usefulness weighting provides an additional 1.20-point improvement within the matched coverage selector. Appendix D provides detailed comparisons, statistical tests, and cost measurements.

6 Conclusion

We introduced RATTS, a reliability-aware approach to *pre-expansion concept allocation* for test-time scaling. Rather than expanding every explored reasoning direction, RATTS allocates compute to concepts that are both useful and complementary through calibrated relevance and semantic coverage. We showed that candidate recall itself is monotone submodular under arbitrary dependence between concept-success events, and connected the practical product-form surrogate to retained candidate recall under an explicit success-mode model. Across three benchmarks and three generators, RATTS preserves nearly all full-expansion candidate recall while substantially reducing inference cost and maintaining comparable final-answer accuracy. More broadly, these results support viewing inference-time reasoning as a resource-allocation problem over candidate directions, rather than only a question of increasing the number of generated solutions. These results suggest that test-time scaling should ask not only *how much* compute to spend, but *which reasoning directions deserve it*.

7 Limitations

The relevance model is supervised using LLM judgments of concept usefulness rather than empirical rollout outcomes. This choice avoids the substantially higher computational cost of generating and verifying multiple solution rollouts for every training concept, but makes the learned relevance signal dependent on the judge's assessment of usefulness. Future work could study rollout-derived or hybrid supervision when larger training-time compute budgets are available.

Reproducibility Statement

We provide the methodological, theoretical, and experimental details needed to reproduce RATTS and its evaluation in the appendix, including the training and selection procedures, evaluation protocol, and statistical analyses. The complete code and associated experimental artifacts will be released upon acceptance.

AI Use Statement

Generative AI tools were used in this work for several supporting tasks. ChatGPT with GPT-5.6 backend was also used to assist with manuscript language editing, formatting, and feedback on experimental presentation and analysis. We additionally used the Stanford Agentic Reviewer (https://paperreview.ai/), CMU Paper Reviewer (Kim et al., 2026b), GPT 6-Astra, and ICLR's LLM Feedback as author-side review tools to obtain preliminary feedback and identify points requiring further clarification or validation.

All AI-assisted suggestions, analyses, and manuscript changes were reviewed by the authors. The authors independently verified the reported experimental results and mathematical claims and take full responsibility for the final technical content, analyses, and conclusions of the paper.

References

Pranjal Aggarwal, Aman Madaan, Yiming Yang, et al. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12375–12396, 2023.

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

Jaime G Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, volume 98, pp. 290941–291025, 1998.

Hyeong Kyu Choi and Sharon Li. Modex: Evaluator-free best-of-n selection for open-ended generation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14394–14416, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Paula Cordero-Encinar and Andrew B Duncan. Certified self-consistency: Statistical guarantees and test-time training for reliable reasoning in llms. *arXiv preprint arXiv:2510.17472*, 2025.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1321–1330. PMLR, 2017.

Divij Handa, Mihir Parmar, Aswin RRV, Md Nayem Uddin, Hamid Palangi, and Chitta Baral. GuidedSampling: Steering LLMs towards diverse candidate solutions at inference-time. In *International Conference on Learning Representations*, 2026.

Jujie He, Tianwen Wei, Rui Yan, Jiacai Liu, Chaojie Wang, Yimeng Gan, Shiwen Tu, Chris Yuhao Liu, Liang Zeng, Xiaokun Wang, Boyang Wang, Yongcong Li, Fuxiang Zhang, Jiacheng Xu, Bo An, Yang Liu, and Yahui Zhou. Skywork-o1 open series, November 2024. URL https://doi.org/10.5281/zenodo.16998085.

Ramya Hebbalaguppe, Ajay Shastry, Soumya Suvra Ghosal, and Chetan Arora. Enhancing deep neural network reliability with refinement and calibration, 2026. URL https://arxiv.org/abs/2605.23249.

Weiyang Huang, Xuefeng Bai, Kehai Chen, Xinyang Chen, Yibin Chen, Weili Guan, and Min Zhang. Sat: Balancing reasoning accuracy and efficiency with stepwise adaptive thinking. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 43384–43402, 2026.

Joongho Kim, Xirui Huang, Zarreen Reza, and Gabriel Grand. Chopping trees: Semantic similarity based dynamic pruning for tree-of-thought reasoning. *arXiv preprint arXiv:2511.08595*, 2025.

Junseok Kim, Nakyeong Yang, Kyungmin Min, and Kyomin Jung. Reliability-aware adaptive self-consistency for efficient sampling in llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 21575–21590, 2026a.

Seungone Kim, Dongkeun Yoon, Kiril Gashteovski, Juyoung Suk, Jinheon Baek, Pranjal Aggarwal, Ian Wu, Viktor Zaverkin, Spase Petkoski, Daniel R Schrider, et al. On the limits and opportunities of ai reviewers: Reviewing the reviews of nature-family papers with 45 expert scientists. *arXiv preprint arXiv:2605.20668*, 2026b.

Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2-3):123–286, 2012.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pp. 510–520, 2011.

Jiayu Liu, Wei Dai, Zhenya Huang, Ning Miao, and Enhong Chen. Verifying large language models' reasoning paths via correlation matrix rank. *arXiv preprint arXiv:2510.24299*, 2025.

Rohin Manvi, Joey Hong, Tim Seyde, Maxime Labonne, Mathias Lechner, and Sergey Levine. Zero-overhead introspection for adaptive test-time compute. In *International Conference on Learning Representations*, 2026.

Jooyoung Moon, Jihyo Kim, Younghak Shin, and Sangheum Hwang. Confidence-aware learning for deep neural networks. In *international conference on machine learning*, pp. 7034–7044. PMLR, 2020.

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

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

Jinming Nian, Zhiyuan Peng, Hongwei Shang, Dae Hoon Park, and Yi Fang. Submodular evaluation subset selection in automatic prompt optimization. *arXiv preprint arXiv:2601.03493*, 2026.

Adam Ousherovitch and Ambuj Tewari. Compute aligned training: Optimizing for test time inference. *arXiv preprint arXiv:2604.24957*, 2026.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Yiming Wang, Pei Zhang, Siyuan Huang, Baosong Yang, Zhuosheng Zhang, Fei Huang, and Rui Wang. Sampling-efficient test-time scaling: Self-estimating the best-of-N sampling in early decoding. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-5415.

Chen Wu, Sachin Goyal, and Aditi Raghunathan. Mode-conditioning unlocks superior test-time compute scaling. In *International Conference on Learning Representations*, 2026.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *International Conference on Learning Representations*, 2025.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. C-pack: Packaged resources to advance general chinese embedding, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023.

Dongxuan Zhu, Ly Tran Ho Khanh, Andy Yat-Ming Cheung, Man-Chung Yue, and Viet Anh Nguyen. Exploring diverse generation paths via inference-time stiefel activation steering. In *International Conference on Learning Representations*, 2026.

Thomas Zollo, Jimmy Wang, and Richard Zemel. Unsupervised confidence calibration for reasoning llms from a single generation. *arXiv preprint arXiv:2604.19444*, 2026.

Jiaru Zou, Ling Yang, Jingwen Gu, Jiahao Qiu, Ke Shen, Jingrui He, and Mengdi Wang. Reasonflux-prm: Trajectory-aware prms for long chain-of-thought reasoning in llms. *Advances in Neural Information Processing Systems*, 38:26764–26802, 2026.

Bowen Zuo and Yinglun Zhu. Strategic scaling of test-time compute: A bandit learning approach. In *International Conference on Learning Representations*, 2026.

APPENDIX CONTENTS

A Extended Related Work **14**

B Proofs **16**

B.1 Proof of Proposition 3.1 16

B.2 Proof of Proposition 3.2 17

B.3 Success-Mode Representation 18

B.4 Proof of Theorem 3.3 19

C Experimental Details **21**

C.1 Prompt Templates and Generation Settings 21

C.2 Concept Supervision and Judge Labeling 26

C.3 Reliability Model and Calibration 27

C.4 Semantic Similarity and Subset Selection 29

C.5 Available Concept Pools and Offline Replay 29

C.6 Answer Aggregation and Correctness Evaluation 32

C.7 Statistical Analysis 33

C.8 LLM-Token Accounting 33

C.9 Selector-Latency Measurement 34

C.10 Metrics and Computational Cost 35

D Additional Results **35**

D.1 Accuracy and Selection Cost 36

D.2 Recall and Accuracy with Reduced Expansion 37

D.3 Performance across Rollout Budgets 40

D.4 Paired Candidate-Recall Comparisons 42

D.5 Refinement and Calibration 42

D.6 Similarity and Selection Ablations 44

A Extended Related Work

Test-time scaling for language-model reasoning. Test-time scaling improves language-model performance by allocating additional computation to generation, search, or candidate evaluation. Self-consistency (Wang et al., 2023) samples multiple reasoning paths and aggregates their answers, while Brown et al. (2024) show that repeated sampling can substantially increase the probability of obtaining at least one correct candidate. The benefit of this computation depends on how it is used. Snell et al. (2025) study allocation based on problem difficulty, and Wu et al. (2025) compare the compute–performance tradeoffs of different model sizes, voting rules, verifier-based selection, and tree search. These studies establish that increasing the number of samples is only one aspect of effective inference-time scaling.

Training and inference can also be designed together. The s1 approach (Muennighoff et al., 2025) combines supervised fine-tuning with budget forcing to control reasoning length, while Compute Aligned Training (Ousherovitch & Tewari, 2026) adapts training objectives to the inference strategy used at deployment. RATTS instead holds the concept and solution generators fixed and learns a separate selector. Its role is to allocate an expansion budget among concepts already proposed for a problem, rather than to change how the underlying generator is trained or to re-establish the relative merits of concept-guided inference versus direct repeated sampling.

Diversity and concept-guided exploration. Structured exploration seeks to avoid repeatedly generating solutions that follow the same strategy. Tree of Thoughts (Yao et al., 2023) explores and evaluates alternative intermediate reasoning states. Mode-conditioning (Wu et al., 2026) distributes computation across reasoning modes using specialist models or mode-specific prefixes, while STARS (Zhu et al., 2026) steers hidden activations to encourage different generation trajectories. These approaches alter how candidate reasoning paths are explored.

Most closely related to our setting, GuidedSampling (Handa et al., 2026) separates inference into concept exploration and concept-conditioned solution generation. The first stage identifies high-level solving directions, and the second uses them to generate candidate solutions. RATTS takes this decomposition as its starting point and formalizes the distinct pre-expansion allocation decision between the two stages: which explored concepts should receive solution-generation compute before any rollout outcome is observed. It retains the available concept pool and chooses which concepts receive expansion compute using estimated usefulness and semantic coverage. Generating a diverse pool therefore does not resolve the allocation problem under a limited expansion budget. RATTS changes neither the exploration procedure nor the underlying solution generator.

Adaptive sampling and compute allocation. Several methods reduce inference cost by deciding when enough responses have been generated. Adaptive-Consistency (Aggarwal et al., 2023) uses agreement among sampled answers to adjust the sampling budget. The Martingale Majority Certificate (MMC) (Cordero-Encinar & Duncan, 2025) provides an anytime-valid stopping rule for identifying the mode of the answer distribution; this certifies agreement with the model's mode rather than correctness of that mode. ReASC (Kim et al., 2026a) combines response frequency with confidence, allowing reliable responses to contribute more strongly to the stopping decision. These methods use information from generated responses to determine whether further sampling is warranted. RATTS instead makes an outcome-free allocation decision over proposed concepts before their solutions are generated.

Adaptive allocation also operates across queries and within partially generated trajectories. Zuo & Zhu (2026) formulate allocation across queries as a bandit problem, directing compute toward difficult but potentially solvable instances. Manvi et al. (2026) predict reward and remaining generation cost during decoding to guide which prefixes should receive more computation. RATTS addresses the distinct, within-problem decision of allocating a given budget across concepts. It does not redistribute a global budget across queries or require intermediate solution feedback to make its selections.

Early termination and semantic pruning. Reducing redundant computation is also a central objective of methods that prune ongoing generations. Self-Truncation Best-of-N (Wang et al., 2025) uses consistency in early hidden states to select a promising path without completing every candidate. Semantic Similarity-Based Dynamic Pruning (SSDP) (Kim et al., 2025) clusters and prunes

semantically redundant steps during tree search. Stepwise Adaptive Thinking (SAT) (Huang et al., 2026) uses process-reward signals to adjust the amount of thinking devoted to individual reasoning steps.

RATTS shares the goal of avoiding unnecessary expansion, but makes its decision over high-level concepts rather than ongoing solution traces. Its soft-coverage rule selects directly from the concept pool without first imposing a hard clustering, and combines semantic overlap with learned usefulness weights. This allows selection to account for both weak directions and repeated variants of useful ones. Concept selection and within-trajectory pruning concern different stages of inference and could be combined, although the present evaluation isolates the concept-selection stage.

Learned verifiers and final-answer selection. Learned verifiers provide quality signals for choosing among generated solutions. Cobbe et al. (2021) train outcome verifiers for mathematical solutions, while Lightman et al. (2024) study supervision at the level of individual reasoning steps. SkyWork's process reward models (He et al., 2024) provide stepwise reward scores, and ReasonFlux-PRM (Zou et al., 2026) uses both step-level and trajectory-level supervision for long reasoning traces. These models motivate learned scoring as an alternative to uniform allocation. RATTS trains a small scorer specifically on judged concept usefulness and combines its predictions with a set-level coverage objective, rather than using individual scores alone to determine the expansion set.

Other selection methods avoid a separately trained verifier. Self-Indicator (Liu et al., 2025) uses the model's internal representations to reweight candidate reasoning paths. ModeX (Choi & Li, 2026) constructs a similarity graph over generated responses and recursively partitions it to identify a representative of the dominant group; ModeX–Lite additionally introduces early pruning. RATTS and ModeX act at different stages: the former selects concepts for generation, while the latter selects a final response from the resulting candidates. We use ModeX as the downstream rule in our concept-selector comparisons and evaluate candidate recall separately from final-answer accuracy.

Refinement and calibration of usefulness estimates. Post-hoc calibration adjusts predicted probabilities using held-out labels (Guo et al., 2017). Confidence-aware learning also studies whether predictions are ordered appropriately by confidence; for example, Moon et al. (2020) introduce Correctness Ranking Loss to encourage higher confidence for correct predictions. Hebbalaguppe et al. (2026) study refinement alongside calibration and propose a joint framework for improving predictive reliability. RATTS applies these ideas to ordinal concept usefulness. Its training objective encourages agreement with usefulness levels and their within-problem ordering, while vector scaling adjusts the resulting probabilities before they are used as coverage weights.

The supervision target distinguishes this scorer from a solution verifier. Concept judgments can be collected without generating and verifying multiple complete rollouts for every training concept. An alternative approach is to derive confidence targets from offline self-consistency and distill them into a lightweight predictor (Zollo et al., 2026). RATTS instead uses judged usefulness available before solution generation. Calibration is therefore defined relative to those labels;the relevance score is not claimed to be a calibrated probability of rollout success. Because the resulting relevance scores are normalized within each problem, absolute probability scale is not itself used by the selector; calibration can affect selection through changes in relative score magnitudes and ordering. The conditional analysis in Section 3 identifies the additional success-model assumptions needed to connect these practical weights to retained candidate recall.

Relevance, diversity, and submodular subset selection. Balancing relevance and redundancy has a long history in information retrieval and summarization. Maximal Marginal Relevance (MMR) (Carbonell & Goldstein, 1998) combines query relevance with a penalty for similarity to previously selected items. Lin & Bilmes (2011) use submodular objectives to combine coverage and diversity in document summarization. More recently, SESS (Nian et al., 2026) selects evaluation examples for automatic prompt optimization through a submodular objective. These methods share the principle that an item's value depends on what has already been selected, but differ in their selection objects and downstream objectives.

Determinantal point processes (DPPs) provide another standard quality–diversity formulation, assigning subsets probability or score through determinants of positive-semidefinite kernels that com-

bine item quality with repulsion between similar items (Kulesza & Taskar, 2012). RATTS instead uses product-form residual coverage. This choice is tied to the inference objective: under the success-mode model in Section 3, retained candidate recall has the same product-sum form as the practical soft-coverage surrogate. The motivation is therefore not simply to apply a generic diversity regularizer.

RATTS applies this principle to concepts generated for an individual reasoning problem. Its relevance-weighted product-form coverage lets additional concepts contribute according to how much weighted mass remains uncovered, without imposing a fixed partition of the concept pool. The greedy approximation guarantee follows the classical analysis of cardinality-constrained submodular maximization (Nemhauser et al., 1978) and is not itself the methodological novelty. Our analysis instead shows that candidate recall itself is monotone submodular under arbitrary dependence between concept-success events. Because this true objective is unavailable before expansion, RATTS uses a practical product-form surrogate; under an explicit success-mode model, retained candidate recall has exactly the same product-sum structure, with parameter mismatch controlling the resulting conditional bound. The contribution is therefore the formulation of pre-expansion concept allocation and the structural connection between its practical surrogate and the candidate-recall objective, rather than a new general-purpose submodular maximization algorithm.

B PROOFS

Throughout this appendix, we fix a problem x, its generated concept pool C_x, and the joint law of the potential rollouts. All probabilities condition on x, C_x, with this conditioning suppressed in the notation. Whenever retained utility is used, we assume $P_x([K_x]) = \mathbb{P}(H_x) > 0$. Empty unions and products are interpreted as the empty event and one, respectively.

B.1 PROOF OF PROPOSITION 3.1

Proof. For $S \subseteq [K_x]$, write

$$E_S = \bigcup_{i \in S} A_i(x).$$

Since $E_S \subseteq H_x$, the definition of conditional probability gives

$$\mathbb{P}(E_S \mid H_x) = \frac{\mathbb{P}(E_S \cap H_x)}{\mathbb{P}(H_x)} = \frac{P_x(S)}{P_x([K_x])} = \mathrm{Ret}_x(S).$$

This proves the conditional-probability representation of retained utility.

The empty set contains no successful concept, so $P_x(\emptyset) = 0$. Moreover, $S \subseteq T$ implies $E_S \subseteq E_T$, and hence $P_x(S) \le P_x(T)$. Thus, P_x is normalized and monotone.

For $j \notin S$, the additional success event obtained by expanding concept j is $A_j(x) \setminus E_S$. Consequently,

$$\begin{aligned} P_x(S \cup \{j\}) - P_x(S) &= \mathbb{P}(A_j(x) \setminus E_S) \\ &= \mathbb{P}\left(A_j(x) \cap \bigcap_{i \in S} A_i(x)^c \right). \end{aligned}$$

If $S \subseteq T$ and $j \notin T$, then

$$A_j(x) \cap \bigcap_{i \in T} A_i(x)^c \subseteq A_j(x) \cap \bigcap_{i \in S} A_i(x)^c.$$

Taking probabilities yields

$$P_x(T \cup \{j\}) - P_x(T) \le P_x(S \cup \{j\}) - P_x(S),$$

which is the diminishing-returns property. The argument uses only event inclusion and therefore permits arbitrary dependence between concept-success events.

Finally, Ret_x is obtained by dividing P_x by the positive constant $P_x([K_x])$. This preserves normalization, monotonicity, and submodularity, while also giving

$$\mathrm{Ret}_x([K_x]) = 1.$$

□

B.2 Proof of Proposition 3.2

Proof. For notational convenience, write $f(S) = \widehat{F}_x(S)$. Recall that

$$f(S) = \sum_{i=1}^{K_x} \mu_i \left[1 - \prod_{j \in S} (1 - \kappa_{ij}) \right],$$

where $\mu_i \geq 0$, $\sum_i \mu_i = 1$, and $\kappa_{ij} \in [0, 1]$.

The empty-product convention gives $f(\emptyset) = 0$. For an unselected concept j, its marginal gain is

$$\begin{aligned}
\Delta_x(j \mid S) &= f(S \cup \{j\}) - f(S) \\
&= \sum_{i=1}^{K_x} \mu_i \left[\prod_{h \in S} (1 - \kappa_{ih}) - (1 - \kappa_{ij}) \prod_{h \in S} (1 - \kappa_{ih}) \right] \\
&= \sum_{i=1}^{K_x} \mu_i \kappa_{ij} \prod_{h \in S} (1 - \kappa_{ih}).
\end{aligned}$$

Every term is nonnegative, so f is monotone. If $S \subseteq T$ and $j \notin T$, then

$$\prod_{h \in T} (1 - \kappa_{ih}) \leq \prod_{h \in S} (1 - \kappa_{ih})$$

for every i, because each additional factor belongs to $[0, 1]$. Multiplying by $\mu_i \kappa_{ij} \geq 0$ and summing gives

$$\Delta_x(j \mid T) \leq \Delta_x(j \mid S).$$

Thus, f is normalized, monotone, and submodular.

We use the following consequence of diminishing returns. For any sets S, T, enumerate

$$T \setminus S = \{j_1, \ldots, j_\ell\}.$$

Adding these elements successively gives

$$\begin{aligned}
f(S \cup T) - f(S) &= \sum_{r=1}^{\ell} \Delta_x(j_r \mid S \cup \{j_1, \ldots, j_{r-1}\}) \\
&\leq \sum_{j \in T \setminus S} \Delta_x(j \mid S). \qquad (5)
\end{aligned}$$

The same inequality holds when $T \setminus S$ is empty.

Let $S^{(t)}$ be the greedy set after t selections, with $S^{(0)} = \emptyset$ and $S^{(B_x)} = \hat{S}_x$. For $t = 0, \ldots, B_x - 1$, monotonicity and equation 5 imply

$$\begin{aligned}
f(S_x^{\mathrm{cov}}) - f(S^{(t)}) &\leq f(S_x^{\mathrm{cov}} \cup S^{(t)}) - f(S^{(t)}) \\
&\leq \sum_{j \in S_x^{\mathrm{cov}} \setminus S^{(t)}} \Delta_x(j \mid S^{(t)}) \\
&\leq B_x \max_{j \notin S^{(t)}} \Delta_x(j \mid S^{(t)}).
\end{aligned}$$

The last inequality uses $|S_x^{\mathrm{cov}}| \leq B_x$. Since greedy chooses a maximum-marginal element,

$$f(S^{(t+1)}) - f(S^{(t)}) \geq \frac{1}{B_x} \left[f(S_x^{\mathrm{cov}}) - f(S^{(t)}) \right].$$

Writing

$$D_t = f(S_x^{\mathrm{cov}}) - f(S^{(t)}),$$

we obtain

$$D_{t+1} \leq \left(1 - \frac{1}{B_x} \right) D_t.$$

Because

$$D_0 = f(S_x^{\mathrm{cov}}),$$

iterating over the B_x greedy selections yields

$$D_{B_x} \le \left(1 - \frac{1}{B_x}\right)^{B_x} f(S_x^{\mathrm{cov}}).$$

Rearranging gives

$$f(\hat{S}_x) \ge \left[1 - \left(1 - \frac{1}{B_x}\right)^{B_x}\right] f(S_x^{\mathrm{cov}}) = \gamma_{B_x} f(S_x^{\mathrm{cov}}).$$

The inequality $1 - u \le e^{-u}$, applied with $u = 1/B_x$, implies

$$\left(1 - \frac{1}{B_x}\right)^{B_x} \le e^{-1},$$

and hence

$$\gamma_{B_x} \ge 1 - \frac{1}{e}.$$

For $B_x = 1$, the expression gives $\gamma_{B_x} = 1$.

It remains to bound the optimization gap after selection. If $\hat{S}_x = [K_x]$, then $\hat{S}_x$ is optimal and the remaining gap is zero. Otherwise, applying equation 5 at $\hat{S}_x$ gives

$$\begin{aligned} f(S_x^{\mathrm{cov}}) - f(\hat{S}_x) &\le f(S_x^{\mathrm{cov}} \cup \hat{S}_x) - f(\hat{S}_x) \\ &\le \sum_{j \in S_x^{\mathrm{cov}} \setminus \hat{S}_x} \Delta_x(j \mid \hat{S}_x). \end{aligned}$$

Let

$$\Delta_x^{(1)}(\hat{S}_x) \ge \Delta_x^{(2)}(\hat{S}_x) \ge \cdots$$

denote the remaining marginal gains of concepts outside $\hat{S}_x$, sorted in decreasing order, and let

$$m_x = \min\{B_x, K_x - |\hat{S}_x|\}.$$

Since

$$|S_x^{\mathrm{cov}} \setminus \hat{S}_x| \le m_x,$$

the preceding inequality gives

$$f(S_x^{\mathrm{cov}}) - f(\hat{S}_x) \le \sum_{k=1}^{m_x} \Delta_x^{(k)}(\hat{S}_x).$$

Finally,

$$\sum_{k=1}^{m_x} \Delta_x^{(k)}(\hat{S}_x) \le m_x \Delta_x^{\max}(\hat{S}_x) \le B_x \Delta_x^{\max}(\hat{S}_x).$$

This establishes the greedy approximation guarantee and both forms of the a posteriori optimization-gap certificate. □

B.3 Success-Mode Representation

We next derive equation 2 under the success-mode assumption.

Proof. Recall the active-mode set

$$\mathcal{I}_x = \{i \in [K_x] : \pi_i > 0\}.$$

For every $i \in \mathcal{I}_x$, the conditional distribution given H_x and $Z_x = i$ is well defined. Conditional independence of the success indicators gives

$$\mathbb{P}\left(\bigcap_{j\in S} A_j(x)^c \,\middle|\, H_x, Z_x = i\right) = \prod_{j\in S} \mathbb{P}(A_j(x)^c \mid H_x, Z_x = i) = \prod_{j\in S}(1 - q_{ij}).$$

Taking the complementary event and applying the law of total probability over Z_x, we obtain

$$\begin{aligned}\mathrm{Ret}_x(S) &= \mathbb{P}\left(\bigcup_{j\in S} A_j(x) \,\middle|\, H_x\right) \\ &= \sum_{i\in\mathcal{I}_x} \pi_i \mathbb{P}\left(\bigcup_{j\in S} A_j(x) \,\middle|\, H_x, Z_x = i\right) \\ &= \sum_{i\in\mathcal{I}_x} \pi_i \left[1 - \prod_{j\in S}(1 - q_{ij})\right],\end{aligned}$$

which is equation 2.

For the full concept set $S = [K_x]$, the factor corresponding to $j = i$ vanishes because $q_{ii} = 1$. Hence every active-mode term equals one, and

$$\mathrm{Ret}_x([K_x]) = \sum_{i\in\mathcal{I}_x} \pi_i = 1,$$

consistent with the definition of retained utility. □

B.4 Proof of Theorem 3.3

Proof. Fix an integer budget $1 \le B_x \le K_x$. For each concept index i, define

$$u_i(S) = 1 - \prod_{j\in S}(1 - \kappa_{ij}).$$

For each active mode $i \in \mathcal{I}_x$, define

$$v_i(S) = 1 - \prod_{j\in S}(1 - q_{ij}).$$

Then

$$\widehat{F}_x(S) = \sum_{i=1}^{K_x} \mu_i u_i(S), \qquad \mathrm{Ret}_x(S) = \sum_{i\in\mathcal{I}_x} \pi_i v_i(S).$$

We first bound their difference at an arbitrary feasible set S. Since $\pi_i = 0$ for $i \notin \mathcal{I}_x$,

$$\widehat{F}_x(S) - \mathrm{Ret}_x(S) = \sum_{i=1}^{K_x} (\mu_i - \pi_i) u_i(S) + \sum_{i\in\mathcal{I}_x} \pi_i \big(u_i(S) - v_i(S)\big).$$

Since μ and π are probability vectors,

$$\sum_i (\mu_i - \pi_i) = 0.$$

As $u_i(S) \in [0, 1]$, centering at $1/2$ therefore yields

$$\begin{aligned}\left|\sum_{i=1}^{K_x} (\mu_i - \pi_i) u_i(S)\right| &= \left|\sum_{i=1}^{K_x} (\mu_i - \pi_i)\left(u_i(S) - \frac{1}{2}\right)\right| \\ &\le \frac{1}{2}\sum_{i=1}^{K_x} |\mu_i - \pi_i| = d_{\mu,x}. \qquad (6)\end{aligned}$$

Thus, the effect of changing the weights is controlled by their total-variation distance.

To control the effect of changing the coverage quantities, consider any

$$a_1, \dots, a_m, b_1, \dots, b_m \in [0, 1].$$

Replacing the factors one at a time gives

$$\prod_{\ell=1}^{m} a_\ell - \prod_{\ell=1}^{m} b_\ell = \sum_{\ell=1}^{m} (a_\ell - b_\ell) \left(\prod_{r=1}^{\ell-1} a_r \right) \left(\prod_{r=\ell+1}^{m} b_r \right).$$

Each product multiplying $a_\ell - b_\ell$ lies in $[0, 1]$, so

$$\left| \prod_{\ell=1}^{m} a_\ell - \prod_{\ell=1}^{m} b_\ell \right| \le \sum_{\ell=1}^{m} |a_\ell - b_\ell|. \tag{7}$$

For $S = \{j_1, \dots, j_m\}$ and any $i \in \mathcal{I}_x$, apply equation 7 with

$$a_\ell = 1 - \kappa_{ij_\ell}, \qquad b_\ell = 1 - q_{ij_\ell}.$$

It follows that

$$|u_i(S) - v_i(S)| \le \sum_{j \in S} |\kappa_{ij} - q_{ij}| \le |S| d_{\kappa,x}. \tag{8}$$

When $S = \emptyset$, both $u_i(S)$ and $v_i(S)$ are zero, so the same bound holds.

Moreover, if $i \in S \cap \mathcal{I}_x$, then

$$\kappa_{ii} = q_{ii} = 1,$$

and hence

$$u_i(S) = v_i(S) = 1.$$

Therefore only active modes outside the selected set can contribute to the coverage discrepancy:

$$\begin{aligned} \sum_{i \in \mathcal{I}_x} \pi_i |u_i(S) - v_i(S)| &= \sum_{i \in \mathcal{I}_x \setminus S} \pi_i |u_i(S) - v_i(S)| \\ &\le |S| d_{\kappa,x} \sum_{i \in \mathcal{I}_x \setminus S} \pi_i \\ &\le |S| d_{\kappa,x}. \end{aligned} \tag{9}$$

The intermediate expression gives the tighter set-dependent form; the last inequality yields the uniform bound used in the theorem.

Combining Eqs. equation 6 and equation 9 gives

$$\begin{aligned} \left| \widehat{F}_x(S) - \mathrm{Ret}_x(S) \right| &\le d_{\mu,x} + |S| d_{\kappa,x} \\ &\le d_{\mu,x} + B_x d_{\kappa,x} \\ &= \eta_x(B_x). \end{aligned}$$

Because this holds for every feasible set,

$$\sup_{|S| \le B_x} \left| \mathrm{Ret}_x(S) - \widehat{F}_x(S) \right| \le \eta_x(B_x). \tag{10}$$

This proves the first claim.

We next transfer the greedy surrogate guarantee to retained utility. Let S_x^{cov} maximize $\widehat{F}_x$ over the feasible sets, while $S_x^\star$ maximizes Ret_x over the same sets. By equation 10, Proposition 3.2, and optimality of S_x^{cov},

$$\begin{aligned} \mathrm{Ret}_x(\hat{S}_x) &\ge \widehat{F}_x(\hat{S}_x) - \eta_x(B_x) \\ &\ge \gamma_{B_x} \widehat{F}_x(S_x^{\mathrm{cov}}) - \eta_x(B_x) \\ &\ge \gamma_{B_x} \widehat{F}_x(S_x^\star) - \eta_x(B_x) \\ &\ge \gamma_{B_x} \left[\mathrm{Ret}_x(S_x^\star) - \eta_x(B_x) \right] - \eta_x(B_x) \\ &= \gamma_{B_x} \mathrm{Ret}_x(S_x^\star) - (1 + \gamma_{B_x}) \eta_x(B_x). \end{aligned}$$

This establishes equation 3.

Finally, applying equation 10 directly at $\hat{S}_x$ gives

$$\mathrm{Ret}_x(\hat{S}_x) \geq \widehat{F}_x(\hat{S}_x) - \eta_x(B_x).$$

Rearranging,

$$1 - \mathrm{Ret}_x(\hat{S}_x) \leq 1 - \widehat{F}_x(\hat{S}_x) + \eta_x(B_x),$$

which is equation 4 and completes the proof. □

C Experimental Details

This appendix continues Section 4 with the prompt templates, supervision procedure, scorer configuration, selection implementation, and evaluation protocol. We also describe the statistical analyses, LLM-token accounting, and selector-latency measurements used to support the results. Detailed performance comparisons are presented in Appendix D.

C.1 Prompt Templates and Generation Settings

We evaluate concept pools with an available-concept target of up to $K_{\max} = 50$, using $M = 5$ solution rollouts per expanded concept. The actual number of available concepts can be smaller than this target. Table 1 gives the generation settings used to construct the stored candidate pools.

Table 1: Generation settings used to construct the candidate pools. The available-concept target is distinct from the number of concepts selected for expansion.

Setting	Value
Maximum available-concept target	$K_{\max} = 50$
Rollouts per expanded concept	$M = 5$
Temperature	0.7
Top-p	0.95
Maximum generation length	8192 tokens

The following templates cover concept generation, concept-conditioned solution and code generation, and usefulness judging. Instance-specific fields are represented by placeholders such as `{problem}`, `{concept}`, `{answer_choices}`, and `{function_signature}`. The templates reproduce no benchmark instance text, generated solution, or evaluation test.

The prompt wording and placeholders are preserved below, with line wrapping and emphasis adjusted for presentation. Box titles and dataset–generator labels are not part of the prompts.

Prompt 1. Mathematical concept generation: first concept
s1K and Math500

You are an expert mathematician. You will be presented with a mathematical question and your task is to identify and state one single, specific theorem or fundamental concept that is most relevant and useful for solving the problem.

QUESTION: `{problem}`

Provide only the name of the theorem or concept, or a concise statement of the principle, that is most directly applicable to solving this problem. Do not attempt to solve the original problem. Only provide the theorem or concept.

Prompt 2. Mathematical concept generation: additional concept
s1K and Math500

You are an expert mathematician. You will be presented with a mathematical question and a list of theorems and concepts that have already been proposed as potentially useful for solving the problem. Your task is to provide a *new* and *different* theorem or concept that is most relevant and useful for solving the problem.

QUESTION: `{problem}`

EXISTING CONCEPTS:
Idea 1: `{concept}`

Provide only the name of the theorem or concept, or a concise statement of the principle, that is most directly applicable to solving this problem. Do not attempt to solve the original problem. Only provide the theorem or concept. If no new, distinct, and useful theorem or concept can be identified, respond with "No additional concepts found."

Prompt 3. Scientific concept generation: first concept
GPQA Diamond

You are an expert science problem solver. You will be presented with a multiple-choice science question and your task is to identify one single, specific concept, principle, mechanism, theorem, or fact pattern that is most relevant and useful for solving the problem.

QUESTION: `{problem}`

CHOICES: `{answer_choices}`

Provide only the name of the concept or a concise statement of the principle. Do not solve the problem. Do not choose an answer. Only provide the concept.

Prompt 4. Scientific concept generation: additional concept
GPQA Diamond

You are an expert science problem solver. You will be presented with a multiple-choice science question and a list of concepts already proposed as potentially useful. Your task is to provide a new and different concept that is most relevant and useful for solving the problem.

QUESTION: `{problem}`

CHOICES: `{answer_choices}`

EXISTING CONCEPTS:
Idea 1: `{concept}`

Provide only the name of the concept or a concise statement of the principle. Do not solve the problem. Do not choose an answer. If no new, distinct, and useful concept can be identified, respond with "No additional concepts found."

Prompt 5. Concept-conditioned solution generation
Math500 | Gemma4-E4B and Qwen3.5-9B

Problem: `{problem}`

Suggested solving concept: `{concept}`

Use the concept if it is helpful. Solve the problem carefully. Put the final answer in `\boxed{}`.

Prompt 6. Concept-conditioned solution generation
Math500 | GLM-4.7-Flash

Problem: `{problem}`

Suggested solving concept: `{concept}`

Use the concept if it is helpful. Solve the problem carefully. Put the final answer in `\boxed{}`.

Response constraints:

- Give only the concise work needed to support the final answer.
- Do not include section labels such as "Reasoning:", "Analysis:", "Thought:", or "Step-by-step reasoning:".
- Do not include hidden reasoning tags.

Prompt 7. Concept-conditioned solution generation
GPQA Diamond | Gemma4-E4B and Qwen3.5-9B

Question: `{problem}`

Choices: `{answer_choices}`

Suggested solving concept: `{concept}`

Use the concept if it is helpful. Choose exactly one option. End your response with:

`Final answer: <A/B/C/D>`

Prompt 8. Concept-conditioned solution generation
GPQA Diamond | GLM-4.7-Flash

Question: `{problem}`

Choices: `{answer_choices}`

Suggested solving concept: `{concept}`

Use the concept if it is helpful. Choose exactly one option. End your response with:

`Final answer: <A/B/C/D>`

Response constraints:

- Give only the concise work needed to support the final answer.
- Do not include section labels such as "Reasoning:", "Analysis:", "Thought:", or "Step-by-step reasoning:".
- Do not include hidden reasoning tags.

Prompt 9. Coding strategy generation: first strategy
HumanEval+

You are proposing high-level coding strategies for a Python function-completion task.

TASK PROMPT:
`{function_signature}`
`{problem}`

Give one concise, specific coding concept or strategy that would help implement the function correctly. Good concepts include algorithm choices, data structures, edge cases, invariants, parsing strategies, and complexity shortcuts.

Return only the concept/strategy, not code and not an explanation.

Prompt 10. Coding strategy generation: additional strategy
HumanEval+

You are proposing high-level coding strategies for a Python function-completion task.

TASK PROMPT:
`{function_signature}`
`{problem}`

EXISTING STRATEGIES:

`- {concept}`

Give one new, distinct, useful coding concept for solving this task. Good concepts include algorithm choices, data structures, edge cases, invariants, parsing strategies, and complexity shortcuts.

Return only the concept/strategy, not code and not an explanation. If no new useful strategy remains, answer:

`No additional concepts found.`

Prompt 11. Concept-conditioned code generation
HumanEval+ | All three generators

You are completing a Python HumanEval+ function.

TASK PROMPT:
`{function_signature}`
`{problem}`

Suggested coding strategy: `{concept}`

Return exactly one complete Python implementation for the same function. Preserve the function name/signature implied by the task prompt. Use the strategy only if helpful. Output raw Python code only: no markdown fences and no explanation.

Prompt 12. Concept-usefulness judging
Fixed semantic concept clusters

You are given a problem and a list of fixed semantic concept clusters.

Your task is to assign a usefulness score to each fixed semantic cluster.

Important:

The clusters are already fixed. Do not split clusters. Do not merge clusters. Do not remove concepts. Do not reassign `member_concept_ids`. Only score each cluster.

For each cluster, judge whether the canonical concept and its member concepts form a useful solving direction for the problem.

Use this score scale:

1.00 = directly useful, central, specific, and actionable. This concept is likely enough to guide a successful solution expansion.

0.75 = useful and relevant, but incomplete, secondary, or not the most direct concept.

0.50 = related, broad, generic, setup-only, or auxiliary. It may help as fallback, but should not be treated as a strong expansion concept.

0.25 = weakly related, likely not useful, computes the wrong object, or too vague.

0.00 = irrelevant, misleading, wrong-domain, vacuous, or clearly harmful.

null = cannot judge.

Scoring rules:

- Do not solve the problem.
- Do not produce the final answer.
- Do not generate a reasoning trace.
- Do not use gold answers.
- Judge only concept usefulness.
- Score the fixed semantic cluster, not individual member concepts.

Prompt 12. Concept-usefulness judging (continued)
Fixed semantic concept clusters

- If the cluster contains a broad concept and a specific concept, score according to the actual fixed cluster and set `diversity_issue` = `mixed_cluster` if appropriate.
- Do not reward generic concepts unless they are truly central.
- Broad labels like "modular arithmetic", "geometry method", "physics principle", "probability counting", "algebraic modeling", "science principle", "formula application", "dimensional analysis", "symmetry", or "counting" should usually be 0.50 unless made specific enough to directly guide this problem.
- A necessary but broad setup concept is usually 0.50 or 0.75, not 1.00.
- A central theorem, formula, mechanism, or decisive solving move for this exact problem can be 1.00.
- A correct but partial substep is usually 0.75.
- A related concept that likely computes the wrong object should be 0.25.
- A wrong-domain or misleading concept should be 0.00.
- Use 0.25 and 0.00 when justified. Do not force all generated concepts to be useful.
- Do not force any specific score distribution.

Return compact JSONL only, one JSON object per input problem.

Output schema:

```
{
  "problem_id": "{problem_id}",
  "cluster_scores": [
    {
      "cluster_index": 0,
      "concept_score": 1.0,
      "concept_score_reason": "{brief_reason}",
      "concept_score_confidence": "high",
      "diversity_issue": "none"
    }
  ]
}
```

Allowed `concept_score` values:

- 1.0
- 0.75
- 0.5
- 0.25
- 0.0
- null

Allowed confidence values:

- high
- medium
- low

Allowed `diversity_issue` values:

- none
- `duplicate_or_paraphrase`
- `too_broad`
- `too_specific`
- irrelevant
- unclear
- `mixed_cluster`

Output restrictions:

- Do not output `member_concepts`.
- Do not echo long concept text.
- `concept_score_reason` must be concise.
- Return JSONL only.

Prompt 12. Concept-usefulness judging (continued)
Fixed semantic concept clusters

- Do not wrap output in markdown.

C.2 Concept Supervision and Judge Labeling

We construct a separate concept-supervision dataset for each generator using `s1K` (Muennighoff et al., 2025). Generated concepts are consolidated into fixed semantic clusters before judging, and the judge assigns a usefulness label to each cluster as a solving direction for the problem. These judgments provide the labeled concept examples used to train the generator-specific scorer. This consolidation belongs to training-data construction; at inference time, RATTS selects directly from the available concept pool using soft coverage.

Concept usefulness was judged using GPT-5.5 Pro through the ChatGPT interface with Extended reasoning enabled. We did not manually set decoding parameters in the interface, and the exact backend decoding settings were therefore not recorded.

The judge receives the problem and the fixed concept clusters, but no gold answer or solution rollouts. Its instructions prohibit solving the problem, returning the final answer, or changing cluster membership. The output records a usefulness score, a brief justification, a confidence category, and a diversity annotation. The five numerical scores form the supervised label space

$$\mathcal{Z} = \{0, 0.25, 0.5, 0.75, 1.0\};$$

`null` denotes an abstention rather than a sixth usefulness class. Table 2 summarizes the scoring rubric.

The training pipelines use BGE-small embeddings and greedy semantic clustering with a cosine-similarity threshold of 0.94. Each cluster has a medoid canonical concept. One labeled cluster contributes one training example consisting of the problem, canonical concept, and cluster score; its member concepts are not expanded into separate training examples. The final training, development, and test files contain no null scores. The training loader requires one of the five numerical labels, and all examples have unit quality weight.

Table 2: Concept-usefulness rubric. The judge scores the fixed concept cluster as a solving direction, without using solution outcomes.

Score	Meaning
1.00	Directly useful, central, specific, and actionable.
0.75	Useful and relevant, but incomplete, secondary, or not the most direct concept.
0.50	Related, broad, generic, setup-only, or auxiliary.
0.25	Weakly related, likely not useful, computes the wrong object, or too vague.
0.00	Irrelevant, misleading, wrong-domain, vacuous, or clearly harmful.
`null`	Cannot judge.

Table 3 reports the numbers of labeled examples in each split. The datasets contain 2,892 examples for `Qwen3.5-9B`, 7,570 for `Gemma4-E4B`, and 6,842 for `GLM-4.7-Flash`. These are counts of labeled concept examples, not distinct problems. The processed datasets contain no malformed final labels.

Table 3: Numbers of labeled concept examples in the supervision splits.

Generator	Train	Development	Test	Total
`Qwen3.5-9B`	2,331	281	280	2,892
`Gemma4-E4B`	5,999	767	804	7,570
`GLM-4.7-Flash`	5,513	654	675	6,842

The splits contain 798 training problems, 99 development problems, and 103 test problems for each generator. All concepts from a problem belong to the same split, and split membership is

identical across generators. Checks of problem IDs and whitespace-normalized question text find no overlap between the training, development, and test sets. The current data refresh preserves the existing assignments for 962 problems and assigns the remaining 38 using the problem-ID SHA256 hash modulo 100: values below 80 enter training, values from 80 to 89 enter development, and the remaining values enter the test split.

Because the supervision concerns concept usefulness, constructing these datasets does not require generating and verifying multiple complete solutions for every training concept. The held-out concept test split is used only for judged-label diagnostics. Benchmark correctness labels and benchmark rollout outcomes are not used to train or calibrate the scorer.

C.3 Reliability Model and Calibration

Each reliability model uses `BAAI/bge-small-en-v1.5` (Xiao et al., 2023) as its encoder backbone. The BGE encoder produces a 384-dimensional pooled representation. We apply dropout with probability 0.1, followed by a learned $384 \to 64$ linear projection and ℓ_2 normalization. A linear $64 \to 5$ classifier then predicts the five usefulness classes. The BGE backbone and the downstream layers are fine-tuned jointly.

All three scorers use the recorded `clean_current` input format and a maximum sequence length of 512. The serialized input is

```
Problem:
{question}

Answer choices:
{choices_text}

Canonical concept:
{canonical_concept}

Generated concept variants:
{variants_text}

Cluster metadata:
cluster_size={cluster_size}
```

Missing answer choices are represented by `(none)`. During benchmark scoring, each available concept is scored independently as a singleton. For a raw concept c_i, the canonical-concept field is the stored concept text, the generated-concept-variants field contains that same concept as a single bullet after whitespace normalization, and `cluster_size=1`. Thus, for a nonempty concept c_i, the corresponding fields are

```
Canonical concept:
{c_i}

Generated concept variants:
- {c_i}

Cluster metadata:
cluster_size=1
```

No semantic clustering or grouping is performed to construct these benchmark-scoring inputs. The same singleton serialization is used for `Qwen3.5-9B`, `Gemma4-E4B`, and `GLM-4.7-Flash`; only the generator-specific scorer checkpoint and resulting predictions differ. The complete serialized string is tokenized jointly and truncated to 512 tokens.

We train with AdamW, a learning rate of 2×10^{-5}, weight decay of 0.01, a warm-up ratio of 0.06, and a gradient-norm limit of 1. Training runs for at most 10 epochs, with early-stopping patience of three epochs.

We train a separate scorer for each generator using the same objective and calibration procedure. Each scorer learns from the concepts produced by its corresponding generator. The shared objective follows Section 2.2:

$$\mathcal{L}_{\text{ref}}(\phi) = \mathcal{L}_{\text{cls}}(\phi) + \lambda_{\text{score}}\mathcal{L}_{\text{score}}(\phi) + \lambda_{\text{rank}}\mathcal{L}_{\text{rank}}(\phi),$$

with

$$\lambda_{\text{score}} = 0.5, \qquad \lambda_{\text{rank}} = 1.0.$$

The classification term is class-weighted cross-entropy over the five usefulness labels. The regression term applies a Huber loss with threshold 0.25 to the difference between the expected usefulness score $\widehat{s}_\phi(x, c)$ and its label z. The ranking term compares expected usefulness scores for concepts with different labels from the same problem, using the pairwise logistic loss defined in Section 2.2.

The training loop combines class-balanced row minibatches of 16 examples with problem-grouped minibatches containing all concepts from eight problems. The within-problem ranking loss is evaluated on the grouped minibatches. Table 4 reports the scorer-evaluation batch sizes and random seeds.

Table 4: Scorer-evaluation batch sizes and training seeds. All three scorers use the same training objective and calibration procedure. Evaluation batch size refers to scorer evaluation, not the latency benchmark.

Generator	Evaluation batch size	Seed
`Qwen3.5-9B`	32	17
`Gemma4-E4B`	64	202607
`GLM-4.7-Flash`	64	202607

Classification weights are computed separately from each training split. If n_z is the number of examples in class z and N is the training-set size, we use

$$u_z = \min\left\{3, \frac{N}{5\max(1, n_z)}\right\}, \qquad w_z = \frac{u_z}{\frac{1}{5}\sum_{v\in\mathcal{Z}} u_v}.$$

Checkpoint selection uses the development split only. Its composite criterion combines macro-F1, quadratic-weighted kappa, maximal-usefulness AUROC, relevant-label AUROC, pairwise accuracy for label gaps of at least 0.5, accuracy on the 1-versus-0.75 boundary, and $1 - \text{ECE}$, with weights $0.20, 0.20, 0.15, 0.15, 0.15, 0.10, 0.05$, respectively. Unavailable components contribute zero, and the checkpoint is updated only on a strict improvement. Each scorer's checkpoint is selected using this criterion on its development split. The selected Qwen, Gemma, and GLM checkpoints are from epochs 7, 6, and 5, respectively. Test outcomes are not used for checkpoint selection.

After training, we hold the scorer parameters ϕ fixed and fit a vector-scaling calibrator on the development split. For logits $h_\phi(x, c) \in \mathbb{R}^{|\mathcal{Z}|}$, the calibrated distribution is

$$\widetilde{p}_{\phi,\eta}(z \mid x, c) = \text{softmax}(a \odot h_\phi(x, c) + b)_z, \qquad \eta = (a, b),$$

where the scales are parameterized as $a = \exp(\ell)$, with $\ell, b \in \mathbb{R}^{|\mathcal{Z}|}$. We fit η by minimizing multiclass negative log-likelihood on raw development logits and judged class labels. The same procedure is applied to all three scorers, with separate calibration parameters fitted for each generator. Other calibrators saved during development are not used for the reported relevance scores.

The relevance score used for selection is

$$p_{\text{rel}}(x, c) = \widetilde{p}_{\phi,\eta}(z = 0.75 \mid x, c) + \widetilde{p}_{\phi,\eta}(z = 1.0 \mid x, c).$$

The two pooled classes both denote concepts judged useful enough to serve as viable expansion directions: score 1.0 denotes a directly useful central direction, while score 0.75 includes useful but incomplete or secondary directions. The resulting quantity therefore estimates judged relevance rather than rollout-success probability.

The scorer and calibrator are fixed before benchmark evaluation. For a given generator, the same pair is applied to `Math500`, `GPQA Diamond`, and `HumanEval+`, without benchmark-specific

training or recalibration. In particular, the programming evaluation uses the scorer trained on `s1K` concept judgments, rather than a programming-specific adapter.

Calibration is evaluated against the held-out judged-usefulness labels. Because p_{rel} is subsequently normalized within each problem before selection, the selector is invariant to multiplying all relevance scores in that problem by a common positive constant. Vector scaling can nevertheless change relative score magnitudes and rankings through its class-specific affine transformation. Thus, improved global calibration metrics should not by themselves be interpreted as the causal source of downstream selection gains. The relevance score is calibrated to judged usefulness and is not claimed to be a calibrated probability of rollout success.

C.4 Semantic Similarity and Subset Selection

We compute semantic similarity using the frozen `BAAI/bge-small-en-v1.5` encoder, separately from the trained reliability scorer. Each stored concept string is tokenized with padding and truncation to a maximum length of 512 tokens. We apply attention-mask-weighted mean pooling to the final token representations and L2-normalize the resulting 384-dimensional vectors in float32. No query prefix, lowercasing, stemming, or additional domain-specific text preprocessing is applied.

For normalized concept embeddings e_i and e_j, we use

$$\kappa_{ij} = \mathrm{clip}\big(e_i^\top e_j, 0, 1\big), \qquad \kappa_{ii} = 1.$$

The replay cache stores these similarities in float16 and reloads them in float32 for selection. We apply no additional temperature or sharpening transformation.

Within each available concept pool, the relevance scores are normalized to give

$$\mu_i = \frac{p_{\mathrm{rel}}(x, c_i)}{\sum_{j=1}^{K_x} p_{\mathrm{rel}}(x, c_j)},$$

with $\mu_i = 1/K_x$ if the denominator is zero. Starting from the empty set, RATTS greedily adds concepts using the marginal gain derived in Section 2.3 until the budget B_x is reached. No hard-clustering or representative-selection stage is applied to the evaluation pool.

The implementation maintains the residual uncovered-mass vector

$$u_i(S) = \prod_{h \in S} (1 - \kappa_{ih}).$$

For an unselected concept j, its marginal gain can therefore be evaluated as

$$\Delta_x(j \mid S) = \sum_{i=1}^{K_x} \mu_i \kappa_{ij} u_i(S).$$

After selecting j_t, the residual state is updated by

$$u_i(S \cup \{j_t\}) = u_i(S)(1 - \kappa_{ij_t}).$$

Given the precomputed $K_x \times K_x$ similarity matrix, evaluating all candidate marginal gains at each greedy step requires $O(K_x^2)$ time. Selection therefore requires $O(B_x K_x^2)$ time and $O(K_x^2)$ space including the similarity matrix. This residual update is the implementation used in the reported selector measurements.

Greedy ties are resolved by higher relevance score and then smaller concept index. The implementation also uses the uniform fallback if the normalizing sum is nonpositive or nonfinite.

C.5 Available Concept Pools and Offline Replay

The evaluation uses 500 `Math500` problems, 198 `GPQA Diamond` problems, and 164 `HumanEval+` problems for each of the three generators. Each available concept has five stored solution rollouts. Methods are evaluated on the same problem, concept, and rollout records, allowing paired comparisons without regenerating candidates for each selection rule.

We preserve the order of valid concepts in the generation records. Empty outputs and responses indicating that no additional concept was found do not enter the valid pool. Valid duplicate concepts remain available to the selector; they are not merged by the training-data consolidation procedure. For an available-prefix target K, we use the first K valid concepts, or all valid concepts if fewer are available. The resulting count is denoted by K_x.

In the concept-count sweep, we use

$$K \in \{1, 2, 3, 5, 8, 10, 15, 20, 30, 40, 50\}$$

and fix the RATTS expansion fraction at $\rho = 0.30$. In the relative-budget sweep, the available target is fixed at $K_{\max} = 50$, and we use

$$\rho \in \{0.01, 0.05, 0.10, 0.20, \ldots, 1.00\}.$$

At either operating point, RATTS selects $B_x = \lceil \rho K_x \rceil$ concepts and retains the five stored rollouts associated with each selected concept. Full Guided Sampling retains all MK_x rollouts. Selection uses only the problem, concepts, scorer outputs, and semantic similarities; no solution rollout, reference answer, or correctness signal is accessed before the expansion set is chosen. In deployment, solution rollouts would be generated only after this pre-expansion selection step.

If method a consumes $N_a(x)$ rollouts on problem x, its plotted rollout consumption is

$$\text{Consumption}_a = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \frac{N_a(x)}{MK_x}.$$

This is the mean of the per-problem fractions, rather than the ratio of total consumed rollouts to total available rollouts. Consequently, the realized consumption of RATTS can exceed the nominal ρ because of integer rounding, particularly when K_x is small.

The concept-count comparisons include Best Score Concepts, Top Cluster Representatives, Random Cluster, SkyWork PRM, ReasonFlux PRM, and the three refinement/calibration variants. At the main operating point $K_{\max} = 50, \rho = 0.30$, we additionally evaluate uniform-weight soft coverage and Maximal Marginal Relevance (MMR). The rollout-budget comparisons also include ASC, MMC, ReASC, and SSDP at their configured budget or stopping settings. Random Cluster results are averaged over three seeds. Each curve uses the method's realized rollout consumption rather than assuming that its control parameter is equivalent to the RATTS expansion fraction.

Uniform-weight soft coverage. To isolate the contribution of the learned relevance weights, we evaluate the same soft-coverage objective with uniform weights

$$\mu_i = \frac{1}{K_x}, \qquad i \in [K_x].$$

The available concept pool, clipped-cosine similarity matrix, expansion budget $B_x = \lceil \rho K_x \rceil$, greedy soft-coverage selection procedure, stored rollouts, and downstream ModeX aggregation are otherwise unchanged. This control therefore retains semantic soft coverage while removing the learned judged-usefulness weighting.

Best Score Concepts. Best Score Concepts uses the same frozen generator-specific scorer, calibrated relevance scores p_{rel}, available concept pool, and expansion budget $B_x = \lceil \rho K_x \rceil$ as RATTS. It selects the B_x highest-scoring concepts and retains the same $M = 5$ stored rollouts per selected concept. Ties are resolved by original concept order. Selection does not penalize semantic overlap, so the selected concepts may describe similar solving directions.

This is the identity-similarity special case of soft coverage. Setting

$$\kappa_{ij} = \mathbf{1}\{i = j\}$$

in Eq. (1) gives

$$\widehat{F}_x(S) = \sum_{i \in S} \mu_i.$$

Since normalization preserves the ordering of the relevance scores, greedy selection then chooses the top B_x concepts by p_{rel}, up to tie-breaking. We therefore use Best Score Concepts as the score-only control rather than reporting a separate identity-similarity variant. As an implementation check, identity soft-coverage selection and direct top-B_x relevance selection produce identical selected concept sets under the same scores and budgets.

Maximal Marginal Relevance. We additionally evaluate Maximal Marginal Relevance (MMR) as a classical relevance–diversity selector. MMR uses the same calibrated relevance scores p_{rel}, clipped-cosine similarities κ_{ij}, available concept pool, and expansion budget $B_x = \lceil \rho K_x \rceil$ as RATTS.

Within each problem, the relevance scores are min–max normalized to $[0, 1]$, giving $\bar{r}_j$. The first selected concept is the concept with highest relevance. For subsequent selections, MMR greedily chooses

$$j_t \in \arg\max_{j \notin S^{(t)}} \left[\lambda \bar{r}_j - (1-\lambda) \max_{h \in S^{(t)}} \kappa_{jh} \right].$$

If all relevance scores within a problem are equal, the normalized relevance term is constant across candidates. Ties are resolved by higher raw p_{rel} and then by smaller original concept index. Each selected concept contributes the same $M = 5$ stored rollouts used by RATTS.

We use $\lambda = 0.50$ for the reported comparison. MMR is evaluated only at the main operating point and is not part of the concept-count or rollout-budget curves.

Cluster-based selectors. Top Cluster Representatives greedily clusters the available concepts using cosine similarity between frozen BGE-small concept embeddings with a threshold of 0.8. Within each cluster, the representative is the concept with the highest calibrated RATTS relevance score. The resulting representatives are ranked by relevance and the top B_x representatives are retained. In our evaluation, the number of clusters was always at least B_x, so no additional budget filling was required.

Random Cluster uses the same clustering procedure, randomly selects B_x clusters, and retains the highest-relevance concept from each selected cluster. Results are averaged over seeds $0, 1, 2$.

PRM selectors. We use `Skywork/Skywork-o1-Open-PRM-Qwen-2.5-1.5B` and `Gen-Verse/ReasonFlux-PRM-1.5B`. Each PRM scores the problem together with one proposed concept, formatted as `Use the following concept or strategy: {concept}`. No generated solution or correctness feedback enters this scoring step. SkyWork applies its input preparation routine and uses the sigmoid value-head reward at the final marked newline. ReasonFlux uses problem–completion concatenation and the sigmoid reward at the last non-padding token. Concepts are ranked independently by descending reward, with ties resolved by original concept order.

These are pre-expansion concept-scoring adaptations of the PRMs rather than verification of completed solutions. They use independent score-based selection and do not combine PRM scores with RATTS soft coverage. Accordingly, comparisons against these baselines jointly reflect differences in the scoring model and the selection rule rather than isolating either component. Timing configurations are given in Appendix C.9.

SSDP. We adapt SSDP to the stored concept-level replay setting by applying semantic pruning to the available concept pool. Concepts are embedded with the frozen `BAAI/bge-small-en-v1.5` encoder using attention-mask-weighted mean pooling and ℓ_2 normalization, and similarity is computed as clipped cosine similarity.

For each similarity threshold, the first currently unassigned concept seeds a group containing the remaining unassigned concepts whose similarity to that seed is at least the threshold. The procedure then continues from the next unassigned concept. We evaluate thresholds

$$\{-1.0, 0.3, 0.5, 0.6, 0.7, 0.75, 0.8, 0.85, 0.9, 0.925, 0.95, 0.975, 0.99\},$$

together with a no-pruning endpoint.

Within each semantic group, the representative is the concept with the largest number of correct stored rollouts, with ties resolved by original concept order. Each retained representative contributes all five stored solution rollouts. Because representative selection uses stored rollout outcomes, this constitutes an offline concept-level replay adaptation of SSDP rather than a deployable pre-expansion selector. We use it as a retrospective semantic-pruning comparator, not as a directly deployable competitor to RATTS. Rollout consumption is the fraction of the full available rollout pool retained after semantic pruning.

Adaptive rollout-budget comparisons. ASC, MMC, and ReASC consume stored complete rollouts in original generation order, up to each problem's full available pool. Malformed outputs count toward consumption. Because the stored Guided Sampling stream is generated in concept-conditioned blocks, these replay adaptations preserve that recorded order rather than recreating an independent unconditioned response stream. Their rollout-budget comparisons should therefore be interpreted as evaluations on the common stored replay pool.

ASC uses the dominance confidence from a Beta distribution with parameters equal to the leading and runner-up answer counts plus one. Its confidence thresholds are $0.5, 0.75, 0.8, 0.9, 0.95, 0.975, 0.99, 0.995, 0.999, 0.9999$, and 0.999999, together with a disabled-stopping endpoint.

MMC uses `PluralityEAndDeciderPrior`, with internal parameter `M`=2, at least one pair update and one other update, and `require_current_leader_match=False`. This internal parameter is distinct from the five rollouts per concept. We vary δ over

$$0.01, 0.05, 0.1, 0.2, \ldots, 0.9.$$

The replay implementation initializes its possible response labels from the full saved response stream and gives each invalid extraction a distinct label; this initialization is specific to the replay setup. Because the label set is initialized from the complete stored stream, the replay can use identities that appear only later in the response sequence. We therefore treat MMC as an offline replay comparator and do not invoke its original anytime-valid online guarantee for this adaptation.

ReASC uses the original generator's full-vocabulary self-certainty, computed from reconstructed prompts and tokenized stored responses. Its confidence statistic uses the bottom 10% of stride-one 128-token windows, including trailing partial windows, with weights $\max\{1, \exp(0.7z)\}$, where z is the confidence value used in the recorded weighting rule. A two-component Gaussian mixture fitted to first-response confidences on the target problem stream provides the high-confidence gate, using a component posterior threshold of 0.95. This is label-free, transductive confidence calibration, separate from the scorer's held-out development calibration. ReASC uses the eleven finite ASC thresholds without a synthetic disabled-stopping endpoint. Confidence preprocessing is not included in the rollout-consumption axis.

C.6 Answer Aggregation and Correctness Evaluation

We use ModeX as the fixed aggregation rule when comparing final-answer accuracy across concept selectors. We use the public ModeX implementation at commit `f1792e9e339c5e08bea761159932b4858598d7ef`, with $\tau = 0.8$ and conductance as the graph-cut criterion. ModeX receives the raw generated response strings, rather than parsed answers or correctness labels.

Its response-similarity graph combines Jaccard similarities over word unigrams, bigrams, and trigrams with weights 0.5, 0.25, and 0.25, respectively. Candidate ordering follows the stored rollout ordering used by the evaluation pipeline, and the same ModeX implementation is used for all concept selectors. Where the ModeX fallback uses randomness, it is seeded deterministically per problem from the dataset, generator, problem identifier, and available-concept target.

ModeX selects from the generated responses without using reference answers or benchmark test outcomes. The selected response is then passed to the task-specific correctness evaluator. The additional-results appendix reports both pass-any and final-answer accuracy so that retaining a correct candidate is distinguished from returning one.

As an implementation consistency check, at $K = 1$ RATTS and full Guided Sampling use the same concept ID, the same five rollout IDs, the same ordered ModeX inputs, and the same final prediction for all $2{,}586$ problem–generator comparisons.

For `Math500`, we extract the final answer and compare it with the reference answer after normalization. For `GPQA Diamond`, we extract the predicted option and compare it with the correct choice. For `HumanEval+`, we extract the generated Python implementation and evaluate it using execution-based tests in a sandboxed environment without network access. These tests are used only for correctness evaluation, not to select concepts or the final program.

Malformed extractions, length-limited generations, and repetition warnings are recorded separately from correctness. The main evaluation retains the benchmark problem set rather than excluding problems on the basis of these warnings. Generation requests remain part of the rollout and token accounting even when their outputs are malformed or incorrect.

C.7 Statistical Analysis

We use paired problem-level bootstrap resampling to quantify uncertainty in method comparisons. Unless otherwise specified below, for pass-any, token-cost, and budget-curve analyses we draw $10{,}000$ samples with replacement within each dataset–generator setting, using a fixed seed. The same sampled problem indices are used for all methods in a comparison, and all outcomes and costs associated with a sampled problem are retained together. We report percentile 95% confidence intervals from the bootstrap distribution.

For Random Cluster, the three seed outcomes are averaged within each problem before bootstrapping. Macro statistics assign equal weight to the nine dataset–generator settings. For these analyses, bootstrap draws are made within each setting and the nine resulting statistics are averaged. This preserves the setting-level resampling procedure used for the reported pass-any, cost, and budget-curve comparisons.

For the endpoint final-answer accuracy comparison between RATTS and full Guided Sampling at $K_{\max} = 50$ and $\rho = 0.30$, we use joint problem-ID resampling across generators. The input consists of $2{,}586$ paired problem–generator records for 862 benchmark problems. Within each benchmark, we sample its original number of problem IDs with replacement, carrying both methods' outcomes and all three generators together. The three benchmarks are sampled independently, and each replicate averages the nine setting-level accuracies equally before taking the paired difference. We use $10{,}000$ resamples with a fixed seed and obtain the interval from the 2.5th and 97.5th percentiles of these differences. This preserves shared-problem dependence across generators for this comparison; it does not change the resampling procedure used for the other reported intervals.

All bootstrap analyses hold the fitted scorers, stored concept and rollout pools, and reported answer-selection settings fixed. Their intervals quantify variation across evaluation problems, not across retraining runs or newly generated rollout pools.

We additionally use exact two-sided McNemar tests for paired binary outcomes at $K_{\max} = 50$ and $\rho = 0.30$, separately for pass-any and ModeX final accuracy. Let n_{10} count problems where RATTS succeeds and the comparator fails, and let n_{01} count problems with the reverse outcome. The exact test uses the binomial distribution with success probability $1/2$, conditional on $n_{10} + n_{01}$. When there are no discordant outcomes, the p-value is one. We apply Holm correction across the 99 comparisons for each metric. Random Cluster seeds are tested separately here, since their mean is not a binary outcome.

For budget-curve comparisons, we compute normalized area under the curve over the observed rollout-consumption interval shared by each pair of methods. Points with identical consumption are averaged before linear interpolation; we then apply trapezoidal integration and divide by the width of the shared interval. Bootstrap resampling retains each problem's complete outcome and cost curves, with the comparison interval held fixed. Different method pairs can therefore have different integration intervals.

The reported confidence intervals and hypothesis tests assess paired performance differences. Failure to detect a statistically significant difference does not establish equivalence or non-inferiority.

C.8 LLM-Token Accounting

We account for the generation requests required by each method using the stored backend input and output token counts. Let $E_x(K)$ denote the exploration tokens required to obtain the available prefix for problem x, including both prompt and output tokens. For a concept selector retaining indices $S_a(x)$, define its expansion cost as

$$G_a(x) = \sum_{i \in S_a(x)} \sum_{m=1}^{M} \left(t^{\text{in}}_{x,i,m} + t^{\text{out}}_{x,i,m} \right).$$

The corresponding dataset totals are

$$T_a^{\text{exp}} = \sum_{x \in \mathcal{D}} G_a(x), \qquad T_a^{\text{expl+exp}} = \sum_{x \in \mathcal{D}} \left[E_x(K) + G_a(x) \right].$$

Full-expansion totals use every available concept in the same prefix. We report the two reductions separately:

$$\text{Reduction}_a^{\text{exp}} = 1 - \frac{T_a^{\text{exp}}}{T_{\text{Full}}^{\text{exp}}}, \qquad \text{Reduction}_a^{\text{expl+exp}} = 1 - \frac{T_a^{\text{expl+exp}}}{T_{\text{Full}}^{\text{expl+exp}}}.$$

Exploration accounting includes valid, duplicate, and rejected attempts through the request that produces the target prefix. If fewer than K valid concepts are available, all recorded exploration attempts are counted. Each selected rollout request is counted once, and repeated prompts contribute input tokens on every request. We retain the backend accounting rather than retokenizing the decoded text.

For the Qwen runs, the stored output lengths include padding within generation batches. The reported reductions therefore quantify token savings under the recorded batching configuration. Padding can differ when selected concepts are expanded in newly formed batches, so these totals should be interpreted as replay-accounted generation costs rather than hardware-independent FLOP measurements.

Local relevance scoring, calibration, embeddings, similarity construction, and answer aggregation are not included in these LLM-generation token totals. Selector overhead is measured separately as latency. The reported reductions describe the generation requests that would be required when only selected concepts are expanded; the full pools were generated in advance for the paired replay experiments.

We first compute token reductions within each dataset–generator setting and then average those percentages equally across the nine settings. This macro-average is distinct from taking a ratio of macro-averaged token totals. Token confidence intervals use the paired problem-level bootstrap procedure described above.

C.9 Selector-Latency Measurement

We measure selector latency on a deterministic sample of 90 problems, with ten problems from each dataset–generator setting. The benchmark uses two warm-up calls followed by three timed repetitions per problem. Models and input files are loaded before timing, and measured model paths recompute their required scores and embeddings rather than reading them from replay caches. CUDA operations are synchronized around the GPU timing regions.

The measurements use an NVIDIA B200 MIG `2g.45gb` allocation. The RATTS relevance scorer runs in float32 on the GPU, and the frozen similarity encoder runs in float32 on the CPU, both with a configured batch size of 128. SkyWork runs primarily in bfloat16 on the GPU, whereas ReasonFlux runs in float32 on the GPU. Both PRMs use batches of at most 64 inputs, with an additional limit of 8,192 padded tokens per batch for ReasonFlux.

For RATTS, the measured interval includes tokenization, relevance scoring and calibration, embedding computation, similarity construction, and greedy subset selection. For the PRMs, it includes input tokenization, model inference, and score-based selection. Guided Sampling measures the corresponding operation of returning the available concept IDs. The other selector baselines include the scoring, embedding, clustering, or selection operations required by their respective implementations.

The mean, median, and 95th-percentile complete-selector latencies are computed over all 270 timed observations for each method (90 problems times three timed repetitions), rather than over 90 per-problem averages. Models are already loaded, and final-answer ModeX aggregation is outside the measured interval.

RATTS uses a compact BGE-small-based scorer (approximately 33M parameters), whereas the evaluated SkyWork and ReasonFlux PRMs are 1.5B-parameter autoregressive models. The implementations also use different batch sizes, numerical precision, and CPU/GPU execution paths. The

measured latency comparison therefore reflects the complete selector implementations used in our evaluation rather than an architecture-controlled comparison of selection algorithms.

The mean complete-selector latency for RATTS is 333.058 ms per problem. Its median is 256.778 ms and its 95th percentile is 708.271 ms. For SkyWork, the corresponding values are 626.940, 261.296, and 2209.329 ms. Thus, the large difference in mean latency relative to SkyWork is primarily associated with its heavier latency tail, while their medians are similar. ReasonFlux has substantially larger latency across the distribution, with mean, median, and 95th percentile of 1886.066, 1869.154, and 3445.770 ms, respectively.

These measurements describe the cost of choosing concepts separately from generation cost. They exclude concept exploration, solution generation, final-answer aggregation, and correctness verification, and are therefore reported separately from the LLM-token reductions.

C.10 Metrics and Computational Cost

Our primary metric is pass-any, the fraction of problems whose selected candidate pool contains at least one correct solution. For method a, let $\mathcal{Y}_a(x)$ denote the candidates retained for problem x. On dataset $\mathcal{D}$,

$$\mathrm{PassAny}_a = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \mathbf{1}\{\exists (c, y) \in \mathcal{Y}_a(x) : R(x, y) = 1\}\,.$$

We report retention relative to full expansion as $\mathrm{PassAny}_a / \mathrm{PassAny}_{\mathrm{Full}}$, evaluated on the same problems and concept pools, and also report final-answer accuracy to distinguish candidate recall from returned-answer correctness.

Token reduction is $1 - T_a / T_{\mathrm{Full}}$, where T_a and T_{Full} are the token totals for method a and full expansion. We report this separately for expansion alone and exploration plus expansion. Totals use backend-recorded input and output tokens, including repeated prompts and recorded batch padding, and therefore reflect replay-accounted generation cost under the recorded batching configuration.

Selector latency is measured separately with models preloaded. For RATTS, it includes relevance scoring, embedding computation, similarity construction, and subset selection, but excludes concept exploration, solution generation, and final-answer aggregation. Macro averages give equal weight to the nine dataset–generator settings.

Unless otherwise specified, paired bootstrap intervals use $10{,}000$ problem-level resamples within each dataset–generator setting, preserving paired outcomes across the methods being compared. The resulting nine setting-level statistics are then averaged equally to obtain each macro replicate. For the endpoint final-answer accuracy comparison between RATTS and full Guided Sampling, we instead resample benchmark problem IDs jointly across all three generators, preserving shared-problem dependence across generators. Appendix C.7 gives the complete statistical protocol, together with exact McNemar tests and Holm correction.

The main figures report pass-any versus available concept count and rollout consumption. The main text summarizes retained pass-any, final-answer accuracy, and computational cost; the appendix provides detailed curves, tables, and measurement procedures.

D Additional Results

We compare candidate recall and final-answer accuracy together with the cost of selecting and expanding concepts. Unless otherwise stated, operating-point results use $K_{\max} = 50$, $\rho = 0.30$, and $M = 5$. Macro averages give equal weight to the nine dataset–generator settings, and Random Cluster results average three seeds. Experimental procedures are given in Appendix C. Table arrows indicate whether higher (↑) or lower (↓) values are better. Bold marks the best point estimates within comparable rows or columns, including ties at the reported precision; it does not indicate statistical significance. Tables of paired differences are not ranked across comparators.

D.1 Accuracy and Selection Cost

At the main operating point, RATTS achieves 95.65% macro-average pass-any, compared with 94.75% for SkyWork PRM and 95.12% for ReasonFlux PRM. The paired difference relative to ReasonFlux is $+0.53$ percentage points with a 95% confidence interval of $[-0.08, 1.15]$, so the recall difference from the strongest PRM comparator is not statistically resolved. Table 5 reports candidate recall and complete-selector latency for each method; the matched final-answer comparison with full Guided Sampling is reported in Table 6.

Table 5: Candidate recall and selection cost at $K_{\max} = 50$. Pass-any averages the nine settings; RATTS uses $\rho = 0.30$. Latencies are measured on 90 problems, ten per setting, with two warm-ups and three repetitions. Models are loaded and caches are not accessed. Guided Sampling's time covers returning all concept IDs, not solution generation. Bold marks the best value in each column, including the full-expansion reference.

	Quality (%)	Complete-selector latency (ms)		
Method	Pass-any ↑	Mean ↓	Median ↓	p95 ↓
RATTS	95.65	333.058	256.778	708.271
Guided Sampling	**96.93**	**0.002**	**0.002**	**0.003**
Best Score Concepts	85.73	342.641	252.456	762.028
Top Cluster Representatives	94.53	239.128	144.692	586.861
Random Cluster	94.61	239.262	144.836	586.996
SkyWork PRM	94.75	626.940	261.296	2,209.329
ReasonFlux PRM	95.12	1,886.066	1,869.154	3,445.770
Calibrated, not refined	94.77	330.660	256.255	692.113
Refined, not calibrated	94.61	330.672	256.524	692.363
No calibration, no refinement	94.64	330.642	256.268	692.058

Best Score Concepts uses the same calibrated relevance scores, available concept pool, and expansion budget as RATTS, but selects the highest-scoring concepts without semantic coverage. RATTS achieves 95.65% macro-average pass-any, compared with 85.73% for this score-only control, 94.53% for Top Cluster Representatives, and 94.61% for Random Cluster. A standard MMR selector using the same relevance scores, semantic similarities, and concept budget reaches 94.55% macro-average pass-any at the pre-specified $\lambda = 0.50$, compared with 95.65% for RATTS. On `HumanEval+` with `Qwen3.5-9B`, RATTS pass-any is 91.46%, versus 71.34% for Best Score Concepts.

The matched final-answer comparison in Table 6 gives 74.42% macro-average accuracy for RATTS and 74.08% for full Guided Sampling. The observed averages are therefore close, although the setting-level differences vary. We report their paired uncertainty together with expansion-token savings in Appendix D.2. Figures 4 and 5 provide enlarged candidate-recall and final-accuracy concept-count curves.

The latency comparison includes the work required to choose concepts, not just the final subset-selection operation. For RATTS, this comprises relevance scoring, embedding computation, similarity construction, and greedy selection; for the PRMs, it comprises tokenization, model inference, and score-based selection. RATTS uses an approximately 33M-parameter encoder-based scorer, whereas the evaluated SkyWork and ReasonFlux PRMs contain 1.5B parameters and use different model architectures, batching configurations, and execution paths. The latency measurements should therefore be interpreted as complete-selector implementation costs rather than an architecture-controlled comparison.

RATTS and SkyWork have similar medians (256.778 ms and 261.296 ms), whereas RATTS has a lower mean (333.058 ms versus 626.940 ms) and substantially smaller 95th percentile (708.271 ms versus $2{,}209.329$ ms). Thus, the large difference in their mean latency is primarily associated with the heavier SkyWork latency tail. ReasonFlux is substantially slower across the measured distribution. Table 7 shows that most RATTS selection time is spent on the scorer and embedding model, with substantially smaller contributions from similarity construction and greedy selection. Stage medians need not sum to the median total. Concept exploration, solution generation, final-

Table 6: Matched final-answer accuracy (%) at $K_{\max} = 50$ and $M = 5$. RATTS uses $\rho = 0.30$, whereas Full denotes full Guided Sampling. Both methods use ModeX. The macro average gives equal weight to the nine settings. Bold marks the higher accuracy in each row, including ties.

	Final-answer accuracy (%) ↑	
Generator	Full	RATTS
Math500		
Gemma4-E4B	82.20	**84.00**
Qwen3.5-9B	92.80	**93.00**
GLM-4.7-Flash	81.60	**82.60**
GPQA Diamond		
Gemma4-E4B	55.56	**56.57**
Qwen3.5-9B	77.78	**79.29**
GLM-4.7-Flash	**59.09**	**59.09**
HumanEval+		
Gemma4-E4B	76.83	**79.27**
Qwen3.5-9B	63.41	**64.02**
GLM-4.7-Flash	**77.44**	71.95
Equal-setting macro	74.08	**74.42**

answer aggregation, and model loading are outside these timings. Device, precision, and batching configurations are given in Appendix C.9.

Table 7: Median stage latency in milliseconds per problem for the complete-selector benchmark. A dash denotes a stage not used by that selector. The stage medians are reported separately and are not added to obtain the median total. Bold marks the lowest value in rows with at least two reported measurements.

	Median stage latency (ms) ↓		
Stage	RATTS	SkyWork PRM	ReasonFlux PRM
Scorer/PRM tokenization	20.540	11.481	**9.926**
Scorer/PRM inference	**73.418**	245.674	1,858.949
Embedding tokenization	4.502	–	–
Embedding inference	156.162	–	–
Similarity construction	0.131	–	–
Subset selection	1.968	**0.024**	0.042

D.2 Recall and Accuracy with Reduced Expansion

Compared with full Guided Sampling, RATTS preserves most candidate recall while substantially reducing generation work. Its macro-average pass-any is 95.65%, versus 96.93% for full expansion, corresponding to approximately 98.7% retained pass-any. At this operating point, RATTS saves 69.23% of expansion tokens and 56.69% of exploration-plus-expansion tokens.

Alongside these savings, RATTS achieves 74.42% macro-average final-answer accuracy, compared with 74.08% for full Guided Sampling. The observed difference is 0.34 percentage points, with a paired 95% bootstrap confidence interval of $[-0.90, 1.62]$. This interval uses $10{,}000$ resamples that retain each benchmark problem's outcomes jointly across both methods and all three generators, as described in Appendix C.7. The interval includes zero, so the average accuracy difference is not statistically resolved; it does not establish equivalence or non-inferiority.

RATTS has higher observed final accuracy in seven settings and matches full expansion in one, as shown in Table 6. The exception is `HumanEval+` with `GLM-4.7-Flash`: RATTS achieves 71.95%, compared with 77.44% for full expansion, a decrease of 5.49 percentage points. This setting uses the mathematics-trained relevance scorer out of domain. Selective expansion also changes the response composition subsequently seen by ModeX, so retaining a correct candidate need not preserve the same consensus answer. The present evaluation does not isolate which of these effects

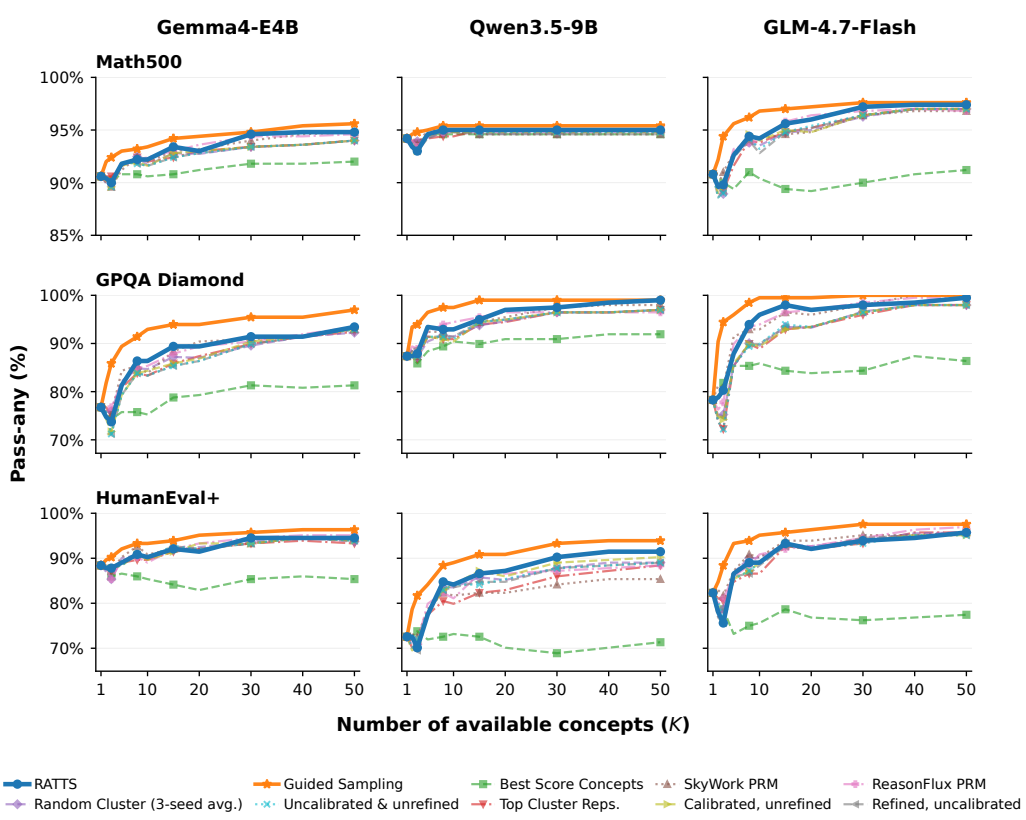


Figure 4: Enlarged pass-any comparison as the available concept pool grows. RATTS expands $B_x = \lceil 0.30 K_x \rceil$ concepts, whereas Guided Sampling expands all K_x concepts. Rows correspond to benchmarks and columns to generators.

accounts for the setting-specific decrease. This motivates reporting candidate recall and returned-answer accuracy separately.

Table 8 gives the setting-level recall comparison. RATTS retains between 96.35% and 100% of full-expansion pass-any. Retention is at least 97.40% on `HumanEval+` across the three generators, using scorers trained on mathematical concept judgments without programming-specific adaptation.

The generation costs in Table 9 quantify the work avoided in these comparisons. RATTS uses an average of 142,710 exploration-plus-expansion tokens per problem, compared with 358,812 for full Guided Sampling; the corresponding 95% bootstrap intervals are $[138{,}833, 146{,}568]$ and $[346{,}756, 370{,}595]$. The totals include backend-recorded input and output tokens, counting repeated prompts on each request. The exploration-plus-expansion reduction accounts for the shared concept-generation stage. Reduction percentages are computed within each setting before averaging, rather than from the ratio of the two macro-average token totals.

These savings are similar to those of most alternative selectors (Table 10), making candidate recall at the reduced generation cost the relevant comparison. For example, Top Cluster Representatives and Random Cluster reduce exploration-plus-expansion tokens by 56.65% and 56.48%, respectively, while achieving 94.53% and 94.61% macro-average pass-any, compared with 95.65% for RATTS. SkyWork and ReasonFlux achieve token reductions of 57.45% and 56.92%, with 94.75% and 95.12% pass-any. As noted above, the observed RATTS–ReasonFlux recall difference is not statistically resolved. Local scoring, embedding, and aggregation computations are not included in the generation-token totals, so token reductions and selector latency remain separate cost measurements within this quality–cost assessment.

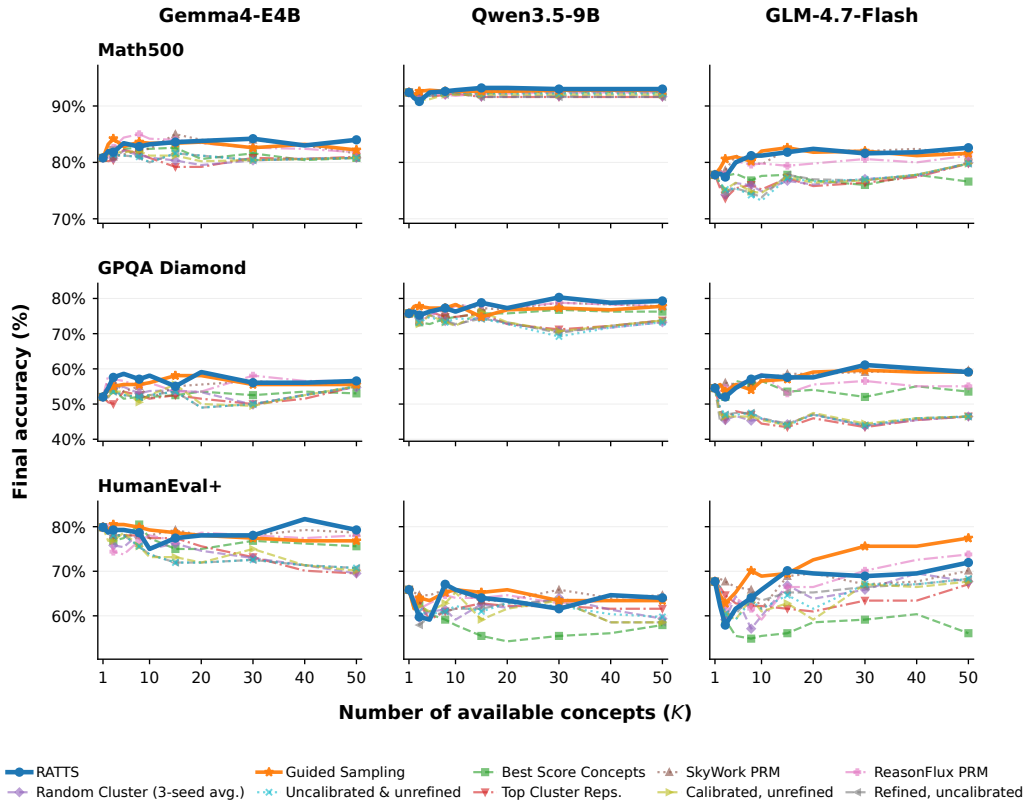


Figure 5: Final-answer accuracy as the number of available concepts varies. RATTS uses $B_x = \lceil 0.30 K_x \rceil$ selected concepts, while Guided Sampling expands the full available pool. Rows correspond to benchmarks and columns to generators. Each selected concept contributes five solution rollouts.

Table 8: Retained pass-any (%) relative to full Guided Sampling at $K_{\max} = 50$. RATTS uses $\rho = 0.30$. Best, Top, and Random denote Best Score Concepts, Top Cluster Representatives, and Random Cluster; SkyWork and ReasonFlux denote the PRM-based concept selectors. These are ratios of pass-any rates, not final-answer accuracies. Bold marks the highest retention in each row, including ties.

	Retained pass-any (%) ↑					
Generator	RATTS	Best	Top	Random	SkyWork	ReasonFlux
Math500						
Gemma4-E4B	**99.16**	96.23	98.33	98.33	**99.16**	98.95
Qwen3.5-9B	**99.58**	99.16	**99.58**	99.37	99.16	99.37
GLM-4.7-Flash	**99.80**	93.44	99.39	99.39	99.18	99.18
GPQA Diamond						
Gemma4-E4B	**96.35**	83.85	95.31	95.14	95.83	**96.35**
Qwen3.5-9B	**100.00**	92.86	97.96	97.96	98.98	97.45
GLM-4.7-Flash	99.49	86.36	97.98	97.98	**100.00**	99.49
HumanEval+						
Gemma4-E4B	98.10	88.61	96.84	97.26	98.10	**98.73**
Qwen3.5-9B	**97.40**	75.97	94.16	94.81	90.91	94.16
GLM-4.7-Flash	98.12	79.37	98.12	98.12	98.12	**99.37**

Table 9: Accounted generation costs at $K_{\max} = 50$, $\rho = 0.30$. Full and RATTS columns report mean exploration-plus-expansion tokens per problem. The final two columns report RATTS token reductions (%) for expansion alone and exploration plus expansion. Bold marks the lower token total in each row; the two reduction columns refer to different accounting boundaries.

Generator	Full tokens ↓	RATTS tokens ↓	Expansion reduction ↑	Exploration-plus-expansion reduction ↑
Math500				
Gemma4-E4B	376,686	**142,547**	70.06	62.16
Qwen3.5-9B	157,240	**65,440**	66.36	58.38
GLM-4.7-Flash	291,310	**119,092**	69.54	59.12
GPQA Diamond				
Gemma4-E4B	562,380	**197,482**	70.02	64.88
Qwen3.5-9B	983,485	**338,391**	68.80	65.59
GLM-4.7-Flash	306,038	**132,491**	69.69	56.71
HumanEval+				
Gemma4-E4B	183,782	**82,086**	69.98	55.33
Qwen3.5-9B	143,699	**79,522**	68.90	44.66
GLM-4.7-Flash	224,684	**127,338**	69.72	43.33

Table 10: Macro-average token reductions (%) with paired 95% bootstrap confidence intervals. Reductions are computed against full Guided Sampling within each setting and then averaged equally. The exploration-plus-expansion measure accounts for LLM generation requests, not all local computation. Bold marks the largest point estimate in each reduction column.

Method	Expansion only ↑	Exploration-plus-expansion ↑
Guided Sampling	0.00 [0.00, 0.00]	0.00 [0.00, 0.00]
RATTS	69.23 [69.03, 69.42]	56.69 [56.30, 57.04]
Best Score Concepts	64.23 [63.89, 64.56]	56.99 [56.44, 57.50]
Top Cluster Representatives	69.26 [69.06, 69.45]	56.65 [56.26, 57.01]
Random Cluster	69.01 [68.82, 69.20]	56.48 [56.09, 56.84]
SkyWork PRM	**70.34** [70.13, 70.55]	**57.45** [57.06, 57.81]
ReasonFlux PRM	69.57 [69.36, 69.77]	56.92 [56.53, 57.28]
Calibrated, not refined	69.03 [68.82, 69.23]	56.50 [56.10, 56.86]
Refined, not calibrated	69.04 [68.84, 69.25]	56.51 [56.12, 56.88]
No calibration, no refinement	69.10 [68.90, 69.29]	56.55 [56.15, 56.92]

D.3 Performance across Rollout Budgets

Figures 6 and 7 report candidate recall and final-answer performance, respectively, across rollout budgets. The paired curve analysis in Table 11 considers both pass-any and final accuracy over the observed rollout-consumption interval shared by each method pair. We integrate over this interval and normalize by its width. These curve summaries are reported separately from the fixed-operating-point accuracy comparison in Table 6.

RATTS achieves higher pass-any AUC than ASC, MMC, ReASC, and the offline SSDP replay adaptation by 3.13–5.87 percentage points, with paired 95% confidence intervals entirely above zero. The final-accuracy comparison is qualitatively different. Relative to ASC, MMC, and ReASC, the corresponding point estimates are -0.89, -0.01, and -0.92 percentage points, respectively, and all three confidence intervals contain zero. RATTS therefore establishes a candidate-recall AUC advantage over these stopping rules, but not a final-accuracy AUC advantage.

This divergence is consistent with the different objectives being measured. Pass-any records whether at least one correct candidate is present, whereas ASC, MMC, and ReASC make stopping decisions from response agreement or confidence, and downstream ModeX depends on the composition and modal structure of the retained responses. Concept selection can therefore preserve or increase candidate diversity without proportionally strengthening the consensus mode used for final-answer aggregation. In addition, these stopping comparisons operate on the common stored

concept-conditioned replay stream described in Appendix C.5, rather than on newly generated unconditioned response streams. The results should therefore be interpreted as replay comparisons of rollout-allocation rules rather than as evidence that one objective dominates the other.

RATTS achieves higher final-accuracy AUC than the offline SSDP replay adaptation by 5.65 percentage points, with a positive paired confidence interval.

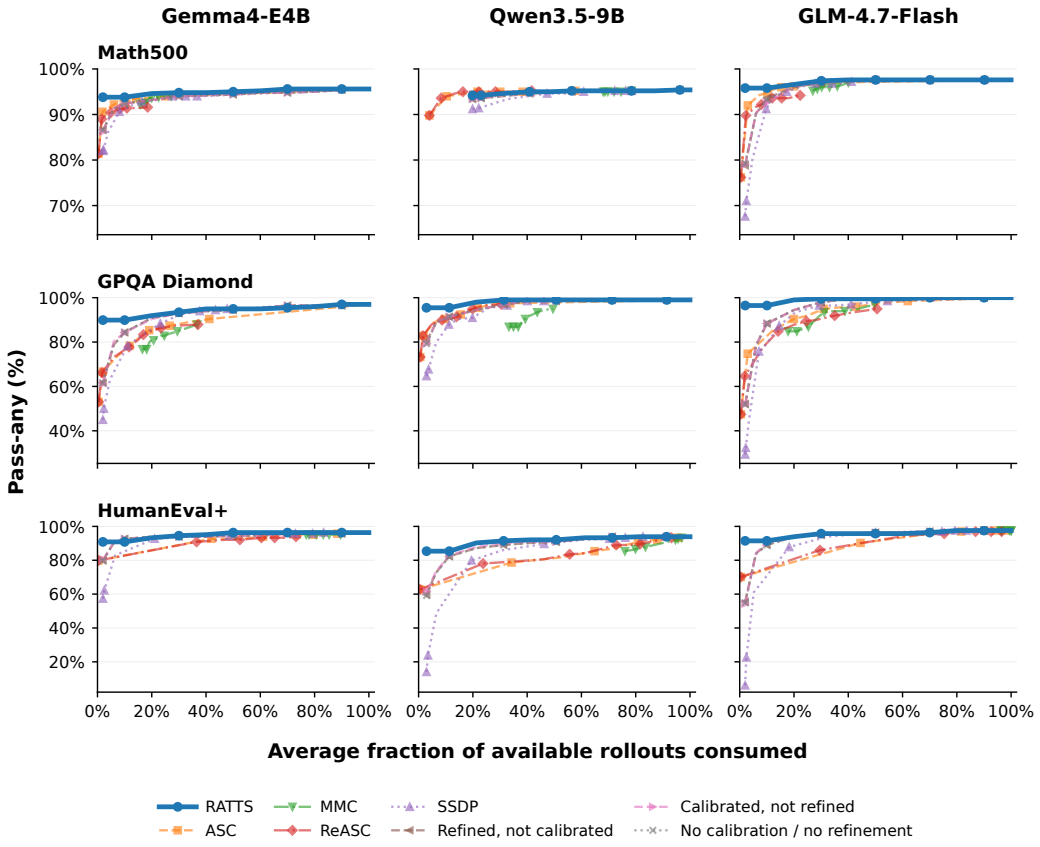


Figure 6: Enlarged pass-any comparison versus the mean per-problem fraction of available rollouts consumed at $K_{\max} = 50$. RATTS varies its expansion fraction ρ, while comparison methods vary their respective budget or stopping parameters. Rows correspond to benchmarks and columns to generators. The horizontal axis measures rollout consumption rather than token cost or latency.

Table 11: Macro-average normalized AUC differences, in percentage points, with paired 95% bootstrap confidence intervals. Each entry is RATTS minus the comparator. AUC is calculated over the fixed observed rollout-consumption interval shared by each method pair, rather than a common $[0, 1]$ interval for all comparisons. Final-accuracy entries summarize the corresponding budget curves. SSDP denotes the offline concept-level replay adaptation described in Appendix C.5.

Comparator	Pass-any AUC ↑	Final-accuracy AUC ↑
ASC	$+3.63$ $[3.06, 4.24]$	-0.89 $[-1.90, 0.13]$
MMC	$+3.73$ $[2.92, 4.56]$	-0.01 $[-1.22, 1.23]$
ReASC	$+5.87$ $[5.01, 6.82]$	-0.92 $[-2.08, 0.20]$
SSDP (offline replay)	$+3.13$ $[2.78, 3.49]$	$+5.65$ $[4.83, 6.49]$
Calibrated, not refined	$+1.27$ $[1.02, 1.54]$	$+3.75$ $[3.03, 4.47]$
Refined, not calibrated	$+1.32$ $[1.07, 1.60]$	$+3.69$ $[2.99, 4.43]$
No calibration, no refinement	$+1.32$ $[1.07, 1.60]$	$+3.67$ $[2.97, 4.40]$

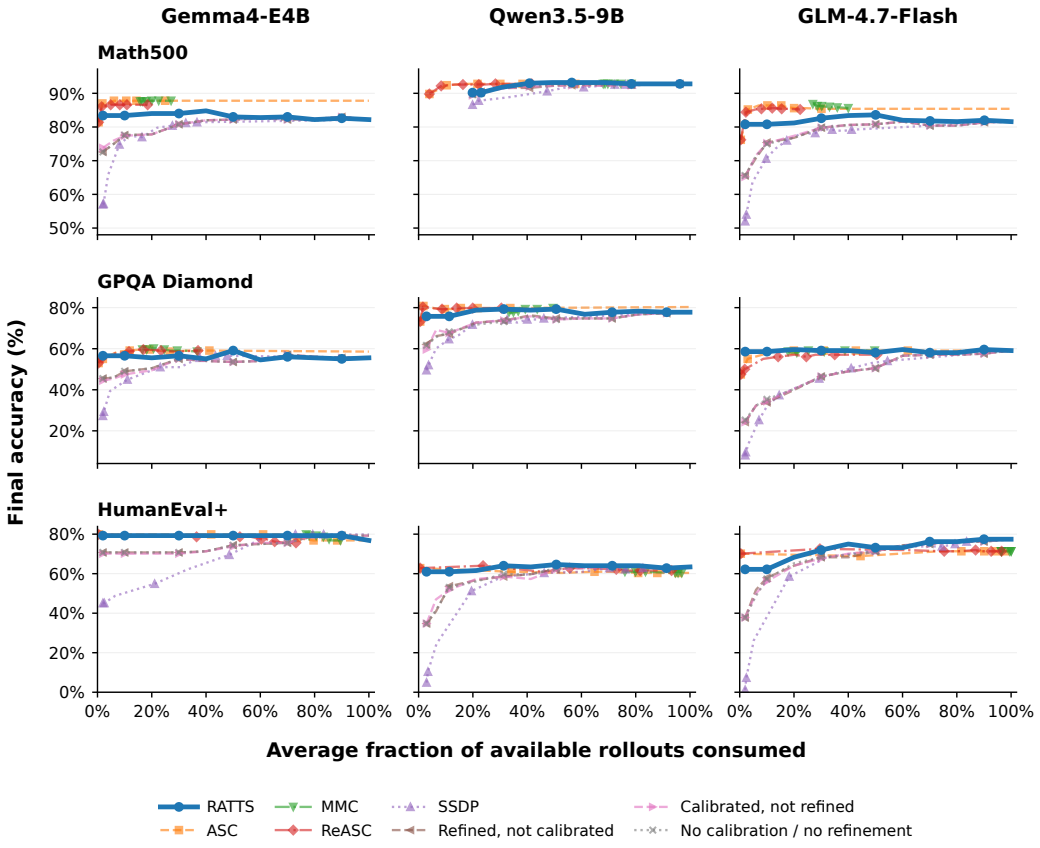


Figure 7: Final-answer accuracy against the mean per-problem fraction of available rollouts consumed at $K_{\max} = 50$. RATTS varies its expansion fraction; comparison methods vary their respective budget or stopping parameters. The horizontal axis measures rollout consumption, not tokens or latency.

D.4 Paired Candidate-Recall Comparisons

The operating-point differences in Table 12 quantify recall comparisons using 10,000 paired problem-level bootstrap resamples. The largest macro-average improvement is over Best Score Concepts: 9.92 percentage points, with a 95% interval of $[8.54, 11.35]$. RATTS has smaller positive differences from the cluster selectors and SkyWork, whose intervals also exclude zero. The difference from ReasonFlux is not statistically resolved. Against full expansion, the -1.28-point difference measures the candidate-recall cost accompanying the generation savings. These bootstrap intervals are not adjusted for multiple comparisons.

Exact two-sided McNemar tests provide complementary setting-level comparisons. After Holm correction across 99 pass-any comparisons, seven differences remain significant at the 0.05 level, all favoring RATTS over Best Score Concepts: the three `GPQA Diamond` settings, the three `HumanEval+` settings, and `Math500` with `GLM-4.7-Flash`. The corrected tests do not establish a difference against either PRM or full Guided Sampling. These tests concern individual settings, whereas Table 12 reports macro-average bootstrap intervals; the two analyses therefore answer different comparison questions.

D.5 Refinement and Calibration

The component comparisons in Tables 12 and 11 evaluate the combined contribution of refinement and vector scaling to the implemented selector. At the main operating point, RATTS improves pass-any by 0.88 percentage points over the calibrated-only variant, 1.03 points over the refined-only

Table 12: Macro-average pass-any differences at $K_{\max} = 50$, in percentage points, with paired 95% bootstrap confidence intervals. Differences are RATTS minus the comparator; positive values favor RATTS.

Comparator	Difference ↑	95% CI
Guided Sampling	-1.28	$[-1.85, -0.79]$
Best Score Concepts	$+9.92$	$[8.54, 11.35]$
Top Cluster Representatives	$+1.11$	$[0.51, 1.73]$
Random Cluster	$+1.04$	$[0.48, 1.62]$
SkyWork PRM	$+0.90$	$[0.22, 1.62]$
ReasonFlux PRM	$+0.53$	$[-0.08, 1.15]$
Calibrated, not refined	$+0.88$	$[0.35, 1.44]$
Refined, not calibrated	$+1.03$	$[0.47, 1.62]$
No calibration, no refinement	$+1.01$	$[0.46, 1.60]$

variant, and 1.01 points over the variant using neither component. All three paired intervals exclude zero. The budget-curve analysis also favors RATTS over these variants for both pass-any and final accuracy. These downstream comparisons measure the effect of the complete scorer variants and do not isolate probability calibration as the causal mechanism.

To examine the scorer on its supervision target, Table 13 reports held-out judged-usefulness metrics. Each entry is the equal-weight mean of the corresponding metric over the three generator-specific held-out scorer test sets. For the refined model, calibration reduces negative log-likelihood from 1.189 to 0.601 and expected calibration error from 0.345 to 0.044. With calibration fixed, refinement increases relevant-label AUROC from 0.931 to 0.945 and pairwise ranking accuracy from 0.884 to 0.906. These results concern prediction of judged concept usefulness, which is the supervision target of the relevance model.

Table 13: Scorer diagnostics on held-out judged concept labels. Each entry is the equal-weight mean over the three generator-specific held-out scorer test sets. NLL denotes negative log-likelihood and ECE denotes expected calibration error. Lower values are better for NLL, Brier score, and ECE; higher values are better for relevant-label AUROC and pairwise ranking accuracy. All entries are on their native scale. Bold marks the best value in each metric column.

Variant	NLL ↓	Brier ↓	ECE ↓	Relevant-label AUROC ↑	Pairwise accuracy ↑
Refined + calibrated	**0.601**	**0.306**	0.044	**0.945**	**0.906**
Refined + uncalibrated	1.189	0.599	0.345	0.932	0.861
Unrefined + calibrated	0.614	0.312	**0.043**	0.931	0.884
Unrefined + uncalibrated	1.167	0.587	0.333	0.924	0.858

Deployment-style scorer serialization. The scorer is trained on canonicalized concept clusters, whereas benchmark-time scoring represents each candidate as a singleton concept. We therefore re-evaluate the three held-out scorer test sets after converting each example to the exact singleton serialization used at inference, while keeping the judged labels, model weights, and calibration parameters fixed. Across the three generator-specific test sets, this change does not degrade judged-usefulness prediction. Relevant-label AUROC changes from 0.945 to 0.948, NLL from 0.601 to 0.596, and Brier score from 0.306 to 0.303; pairwise ranking accuracy remains essentially unchanged (0.906 versus 0.904). Thus, the benchmark-time singleton representation preserves the scorer's held-out performance on its supervision target.

The downstream ablations assess the combined effect of refinement and vector scaling on selection. Since relevance weights are normalized within each problem, multiplying all relevance scores by the same positive constant leaves selection unchanged. Vector scaling can change relative score magnitudes and rankings through its class-specific transformation, so downstream gains do not isolate improved global probability calibration as their cause.

Table 14: Held-out judged-usefulness performance under the original training-style serialization and the singleton serialization used at benchmark time. Each entry is the equal-weight mean over the three generator-specific scorer test sets. The same 1,759 labeled examples are evaluated in both conditions; only the input serialization changes.

Representation	NLL ↓	Brier ↓	ECE ↓	Relevant-label AUROC ↑	Pairwise accuracy ↑
Original	0.601	0.306	**0.044**	0.945	**0.906**
Singleton	**0.596**	**0.303**	0.045	**0.948**	0.904

D.6 Similarity and Selection Ablations

We first compare the production similarity matrix with rescaled cosine and TF–IDF cosine while keeping the scorer and expansion budget fixed. The production matrix uses clipped cosine,

$$\kappa_{ij} = \mathrm{clip}(e_i^\top e_j, 0, 1), \qquad \kappa_{ii} = 1.$$

Rescaled cosine instead uses

$$\kappa_{ij}^{\mathrm{rescaled}} = \mathrm{clip}\left(\frac{e_i^\top e_j + 1}{2}, 0, 1\right), \qquad \kappa_{ii}^{\mathrm{rescaled}} = 1.$$

For TF–IDF cosine, we use lowercase word unigrams, raw term frequencies, smoothed inverse-document frequency, and ℓ_2-normalized vectors. The TF–IDF vocabulary is fitted separately to the available concept pool for each problem, and the diagonal is set to one. An independently reconstructed clipped-cosine matrix reproduces the production selections and outcomes exactly, confirming the stated similarity implementation.

Contribution of semantic coverage and relevance weighting. The component comparisons distinguish independent relevance ranking, semantic coverage without learned weights, and full RATTS at the same concept budget. Best Score Concepts, which ranks concepts independently using the learned relevance scores, achieves 85.73% macro-average pass-any. Uniform-weight soft coverage reaches 94.45%, showing that coverage alone performs substantially better than independent score ranking in this evaluation. Full RATTS reaches 95.65%.

The matched comparison between uniform-weight coverage and RATTS isolates the contribution of learned relevance weighting within the same soft-coverage selector: RATTS improves pass-any by 1.20 percentage points, with a paired 95% confidence interval of $[0.91, 1.52]$. Thus, the observed progression is

$$85.73\% \quad \longrightarrow \quad 94.45\% \quad \longrightarrow \quad 95.65\%$$

for independent score ranking, uniform semantic coverage, and full RATTS, respectively. Because the first comparison changes both the weighting scheme and the selection structure, it should not be interpreted as a causal decomposition of their individual contributions. The matched uniform-weight comparison nevertheless shows that learned judged-usefulness weights provide an additional improvement within the same semantic-coverage objective.

For final accuracy within the same fixed-scorer replay, RATTS exceeds uniform-weight coverage by 1.25 percentage points, with a 95% confidence interval of $[-0.02, 1.52]$. Because this interval includes zero, the final-accuracy difference is not statistically resolved.

Comparison with Maximal Marginal Relevance. We also compare with Maximal Marginal Relevance (MMR), using the same calibrated relevance scores, clipped-cosine similarities, available concept pools, and expansion budgets as RATTS. We evaluate

$$\lambda \in \{0.25, 0.50, 0.75\},$$

obtaining macro-average pass-any of 94.52%, 94.55%, and 94.50%, respectively. We use the pre-specified $\lambda = 0.50$ as the primary comparison. At this setting, RATTS achieves 95.65% macro-average pass-any, an observed difference of approximately 1.10 percentage points. The similar MMR performance across the three tradeoff values indicates that this point estimate is not driven

by a particular λ choice. We do not attach inferential significance to the observed MMR difference here.

MMR penalizes similarity only to the most similar already selected item through an explicit relevance–redundancy tradeoff. RATTS instead accumulates residual coverage of the full weighted concept pool through the product-form objective. The comparison therefore tests a standard relevance–diversity heuristic using the same underlying relevance and similarity information.

Similarity sensitivity. Table 15 reports paired differences from the production similarity configuration within the fixed-scorer sensitivity replay. These differences are evaluated within that replay and should not be read as a second absolute final-accuracy endpoint; the matched operating-point final accuracy is reported in Table 6.

Rescaling cosine leaves pass-any unchanged and produces a small final-accuracy difference whose interval includes zero. TF–IDF yields a slightly lower pass-any point estimate and a slightly higher final-accuracy point estimate, with both confidence intervals including zero. These comparisons therefore do not establish a measurable advantage of the dense production similarity over the alternatives tested here.

The TF–IDF result is also relevant to deployment cost. Dense embedding inference constitutes a substantial fraction of RATTS selector latency (Table 7), whereas TF–IDF does not require the frozen neural embedding forward pass. We do not report a separately timed TF–IDF selector, so no latency improvement is claimed; the result nevertheless suggests that sparse similarity may be a useful engineering alternative when selector latency is more important than using dense semantic embeddings.

Table 15: Similarity sensitivity at $K_{\max} = 50$, $\rho = 0.30$. Entries are variant minus production, in percentage points, with paired 95% bootstrap confidence intervals. All comparisons are computed within the same fixed-scorer sensitivity replay.

Similarity variant	Pass-any difference ↑	Final-accuracy difference ↑
Clipped cosine	$0.00\ [0.00, 0.00]$	$0.00\ [0.00, 0.00]$
Rescaled cosine	$0.00\ [0.00, 0.00]$	$-0.03\ [-0.24, 0.16]$
TF–IDF cosine	$-0.30\ [-0.81, 0.19]$	$+0.16\ [-1.09, 1.43]$