Adaptive Test-Time Compute Allocation with Evolving In-Context Demonstrations

Bowen Zuo[1] , **Dongruo Zhou**[2] , **Yinglun Zhu**[1]
[1]University of California, Riverside
[2]Indiana University Bloomington
bzuo002@ucr.edu, dz13@iu.edu, yzhu@ucr.edu

Abstract

While scaling test-time compute can substantially improve model performance, existing approaches either rely on static compute allocation or sample from fixed generation distributions. In this work, we introduce a test-time compute allocation framework that jointly adapts where computation is spent and how generation is performed. Our method begins with a warm-up phase that identifies easy queries and assembles an initial pool of question-response pairs from the test set itself. An adaptive phase then concentrates further computation on unresolved queries while reshaping their generation distributions through evolving in-context demonstrations—conditioning each generation on successful responses from semantically related queries rather than resampling from a fixed distribution. Experiments across math, coding, and reasoning benchmarks demonstrate that our approach consistently outperforms existing baselines while consuming substantially less inference-time compute.

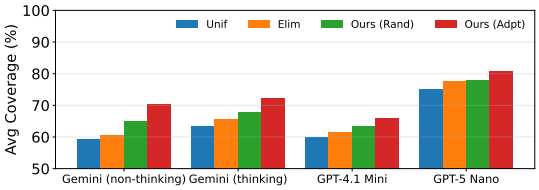


Figure 1: Average coverage (%) across 4 model variants, where each result is averaged over 5 benchmarks spanning math, coding, and reasoning. See Section 5 for details.

1 Introduction

Test-time strategies (Wei et al., 2022; Yao et al., 2023; Madaan et al., 2023) have long been a central focus in the study of Large Language Models (LLMs). In recent years, test-time scaling (Agarwal et al., 2024; Muennighoff et al., 2025; Liu et al., 2025; Snell et al., 2025; Brown et al., 2024) has emerged as an effective alternative to traditional training-time scaling (Chowdhery et al., 2023; Hoffmann et al., 2022; Kaplan et al., 2020). Rather than increasing model size or training data, test-time scaling improves performance by allocating additional computation at inference time, typically by generating multiple candidate responses and selecting the most promising one. Representative approaches include Best-of-N sampling (Brown et al., 2024; Snell et al., 2025), self-consistency (Wang et al., 2023), and reward-guided selection using learned or ground-truth verifiers (Zhang et al., 2025; Lightman et al., 2024; Cobbe et al., 2021; Uesato et al., 2022). More recently, reasoning-oriented models (OpenAI, 2024; DeepSeek-AI et al., 2025) have further amplified the benefits of test-time computation by enabling multi-step deliberation.

Despite this progress, most existing test-time scaling approaches share a common limitation: they rely on a largely static generation process and allocate samples uniformly across test questions. Representative examples include Best-of-N methods (Snell et al., 2025; Brown et al., 2024). While such approaches are simple to implement and admit favorable theoretical guarantees, their lack of adaptivity makes them suboptimal across heterogeneous datasets. A more flexible line of work extends this paradigm by allocating a question-specific number of samples, typically based on estimated question difficulty (Wang et al., 2025b; Damani et al., 2025; Zuo and Zhu, 2026). Even in these methods, however, the generation distribution for each question remains static and fixed throughout the whole generation process. A separate line of work instead modifies the sampling behavior itself (Liu et al., 2025; Lin et al., 2025; Wu et al., 2025; Tuyls et al., 2026), but these meth-

Findings of the Association for Computational Linguistics: ACL 2026, pages 35156–35173

ods typically require running a dedicated search procedure for each individual query or auxiliary exploration mechanisms, adding inference-time overhead and complicating deployment.

In this work, we propose a unified view of *adaptive test-time compute allocation* that jointly considers *how much compute to allocate* and *how that compute is used to improve the sampling distribution* at inference. Our key insight is that effective test-time scaling should not only decide *where* to spend computation, but also *how* additional computation influences the model's conditional generation behavior. We achieve this through self-adaptive in-context learning (ICL), which reuses previously generated responses as contextual demonstrations, allowing subsequent generations to adapt in a lightweight manner without complicated optimization procedures.

Concretely, our method proceeds in two stages. A lightweight warm-up phase allocates a small, fixed budget to every query, resolving easy cases early and assembling an initial pool of question-response pairs from the test set itself. An adaptive phase then concentrates further computation on unresolved queries and, rather than resampling from the base distribution, conditions each generation on demonstrations retrieved from the evolving pool via semantic similarity. As more queries are resolved, the pool grows and the effective sampling distribution shifts—unifying compute allocation and distributional adaptation within a single inference-time loop.

Empirically, our method yields consistent improvements across a range of challenging reasoning benchmarks, including MATH-500 (Lightman et al., 2024), GPQA-Diamond (Rein et al., 2024), LiveCodeBench (Jain et al., 2025), and MinervaMath (Lewkowycz et al., 2022). Across multiple random seeds and evaluation rounds, our approach consistently outperforms a uniform allocation baseline and an adaptive allocation baseline, achieving higher performance at lower compute budgets. Notably, the gains emerge early during test-time scaling, indicating more efficient allocation of computation rather than reliance on late-stage oversampling.

2 Related Work

Test-time scaling. A growing line of research investigates how additional computation at inference time can improve the performance of large language models, without modifying their parameters. This paradigm—often referred to as test-time compute or test-time scaling—has shown that the use of inference-time resources can close or even surpass the gap between prompting-based methods and finetuning (Brown et al., 2020; Mosbach et al., 2023), and that encouraging longer deliberation or "thinking" at inference time yields consistent gains on reasoning benchmarks (Muennighoff et al., 2025; DeepSeek-AI et al., 2025). Among these approaches, repeated sampling has proven particularly effective: methods such as Best-of-N generate multiple candidate responses and rely on reward models or verifiers to identify high-quality outputs (Brown et al., 2024; Snell et al., 2025; Cobbe et al., 2021; Lightman et al., 2024; Liu et al., 2025). Subsequent variants further modulate the generation process based on intermediate signals such as model confidence or partial correctness (Sun et al., 2024; Manvi et al., 2024; Tan et al., 2025). While effective, these methods typically operate at the level of individual queries, adjusting inference effort locally without coordinating computation across inputs.

Test-time compute allocation. A separate but related direction considers how test-time computation should be distributed across an entire set of queries, avoiding over-investment in easy instances while reserving resources for harder ones. While test-time compute allocation has been demonstrated effective, most existing work rely on staged pipelines, auxiliary predictors, or pre-computed signals to guide allocation decisions (Damani et al., 2025; Wang et al., 2025b). One recent work (Zuo and Zhu, 2026) formalizes test-time compute allocation as pure-exploration-style bandit learning (Bubeck et al., 2009; Jamieson and Nowak, 2014; Locatelli et al., 2016; Zhu et al., 2020, 2021, 2022), which enables fully adaptive compute allocation without substantial overhead. However, to our knowledge, all existing allocation methods (Damani et al., 2025; Wang et al., 2025b; Zuo and Zhu, 2026) sample from a static generation distribution for each question and are not adaptive in terms of *how* individual generation is performed. Our work builds on this line of research by coordinating computation across queries while also allowing inference-time generation behavior to evolve during scaling.

In-context learning. In-context learning (ICL) enables language models to adapt their behavior at

inference time by conditioning on demonstrations provided in the prompt, without any parameter updates (Brown et al., 2020; Agarwal et al., 2024; Bertsch et al., 2025). Demonstrations are typically selected via similarity-based retrieval from a labeled set (Liu et al., 2021; Xu et al., 2023; Tanwar et al., 2023), and a consistent finding is that semantically relevant examples yield more reliable performance gains (Yoran et al., 2024). Beyond retrieval from fixed datasets, recent work explores constructing demonstrations dynamically from model outputs themselves (Madaan et al., 2023; Qin et al., 2024; Acikgoz et al., 2025). Xia et al. (2025) study ICL specifically in the test-time scaling regime, but focus on individual queries in isolation. In this paper, we leverage ICL as a lightweight mechanism for reshaping the generation distribution within an adaptive test-time compute allocation framework—using successfully solved test queries as evolving demonstrations to guide inference on remaining ones.

3 Problem Setting

We study how to adaptively allocate inference-time compute across test questions based on their estimated difficulty, with the goal of improving performance over static compute allocation schemes that ignore variation in question difficulty.

We begin with the simplest setting, in which test-time computation is used solely for response generation. Given a test question $x \in \mathcal{X}$, the language model defines a fixed conditional distribution $p(\cdot \mid x)$. Applying additional computation to x corresponds to drawing additional samples from this distribution. Formally, let $c(x_i) \in \mathbb{N}$ denote the number of responses generated for test question x_i up to a given point in the inference process. The corresponding response set is

$$g(x_i; c(x_i)) = \{y_1, \ldots, y_{c(x_i)}\}, y_j \sim p(\cdot \mid x_i).$$

Rather than fixing $c(x_i)$ in advance, adaptive allocation determines these values *progressively*, based on intermediate feedback from previously generated responses. The goal is to concentrate computation on unresolved or difficult questions while avoiding unnecessary sampling for questions that can be answered confidently with fewer generations.

We evaluate performance using *coverage* (Brown et al., 2024; Zuo and Zhu, 2026). A test question is considered correctly answered if at least one generated response is correct:

$$\mathsf{Coverage}(x_i; c(x_i)) = \mathbb{I}\Big\{\exists\, y \in g(x_i; c(x_i)) : y \text{ correctly answers } x_i\Big\}.$$

From this perspective, test-time compute allocation can be viewed as a sequential decision process that repeatedly selects which questions to allocate additional computation to, until all questions are resolved or further computation becomes unproductive. Uniform strategies that allocate the same number of samples to every question ignore differences in difficulty and often waste computation. In contrast, adaptive allocation aims to resolve easy questions early and reserve additional computation for harder cases, achieving strong performance with substantially lower overall compute.

4 Methods

4.1 Intuition

Recent studies suggest that increasing diversity in generated responses is an important factor in effective test-time scaling (Lin et al., 2025; Wu et al., 2025). We view this effect as a consequence of how test-time interventions reshape the model's sampling distribution. However, most existing adaptive sampling methods focus primarily on *where* to allocate compute—e.g., concentrating samples on harder queries—while continuing to draw samples from a fixed conditional distribution. As a result, additional sampling often yields diminishing returns when correct solutions have low probability under the base distribution.

One natural approach to address this limitation is to modify the sampling distribution itself. In-context learning (ICL) provides a flexible mechanism for inducing such shifts by conditioning generation on additional examples. However, prompting is not always beneficial: added context can degrade performance due to context-length effects, demonstration bias, or poorly matched examples (Yoran et al., 2024; Du et al., 2025). A consistent finding in the ICL literature is that similarity-based demonstration selection improves reliability. Methods such as kNN prompting and retrieval-augmented ICL show that conditioning on semantically related examples reduces noise and leads to more stable performance gains (Liu et al., 2021; Tanwar et al., 2023; Xu et al., 2023; Yoran et al., 2024).

Taken together, these findings suggest that the effectiveness of test-time interventions depends

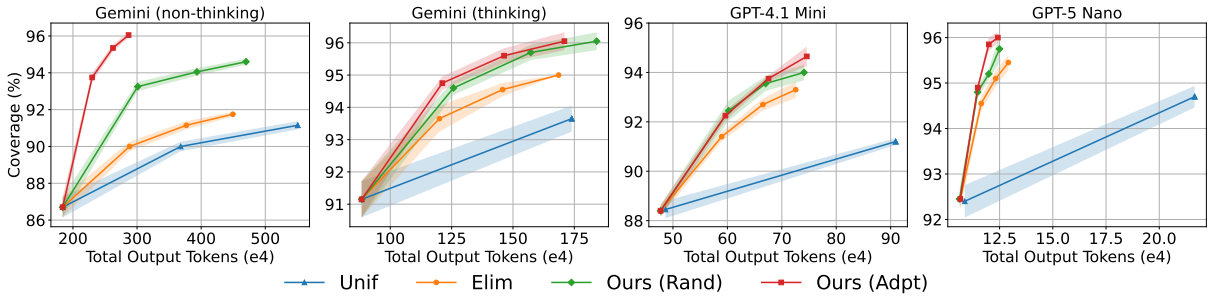


Figure 2: Coverage comparison on MATH-500.

#Samples/q	1	2	3	4	5	6	7	8
Fixed	65.15	72.73	77.78	79.80	81.31	82.32	83.84	85.86
Neigh	65.15	75.76	80.30	81.31	82.83	85.86	86.36	87.88

Table 1: Coverage performance vs. samples per question.

not only on how compute is allocated, but also on whether the induced distribution shift is well aligned with the target query. To illustrate this effect, we conduct a controlled experiment on the GPQA-Diamond dataset, where each question is evaluated under an identical sampling budget using either a fixed prompt or a prompt constructed from semantically neighboring questions. As shown in Table 1, neighbor-based prompting consistently yields higher coverage, indicating that similarity-guided distribution shifts enable more effective use of test-time compute.

4.2 Our algorithm

Given a set of test questions $\mathcal{S} = \{x_1, \ldots, x_n\}$, our goal is to improve test-time performance by applying an adaptive scaling strategy. Our approach leverages prior test-time scaling methods and retrieval-based methods by continuously reusing generated and evaluated test-time samples to update in-context prompts, enabling the conditional response distribution to evolve over time rather than remain fixed. This evolving mechanism operates entirely at inference time, without modifying model parameters.

Our method proceeds in two stages:

1. A *Warm-up stage*, which applies a lightweight, fixed distribution sampling strategy to identify easy queries and construct an initial D_{test} of candidate demonstrations.

2. An *Adaptive Distribution Sampling stage*, which focuses more computation on unresolved queries and conditions future generations on evolving D_{test}.

This two-stage design enables the model to concentrate inference on challenging queries while actively reshaping the response distribution through evolving in-context demonstrations. The full procedure is summarized in Algorithm 1. The framework is modular and can accommodate alternative test-time sampling interventions, demonstration selection strategies, and aggregation rules.

4.2.1 Warm-up

In the warm-up stage, we allocate a small, fixed amount of compute to each query. For each $x \in \mathcal{S}$, we generate a fixed number of responses

$$g_0(x) = \{y_1, \ldots, y_K\}, \qquad y_i \sim p(\cdot \mid x).$$

Each response is evaluated by a reward oracle $r(x, y)$ usually with a range $[0, 1]$. If any response satisfies $r(x, y) \geq \gamma$, then the question is marked as solved and removed from further consideration. Given that our evaluation metric is *coverage*, we use ground-truth as our reward model r to evaluate the correctness directly for this step and a $\gamma = 1$ correspondingly. The warm-up stage serves two purposes. First, it resolves easy queries early, preventing unnecessary compute expenditure, including ICL-construction, which increases input token usage. Second, it constructs an initial D_{test} that will later serve as candidates for ICL demonstrations. This stage also naturally serves as a difficulty estimation process over the test set.

4.2.2 Adaptive Distribution Sampling

After the warm-up stage, the remaining test questions correspond to instances for which correct responses have low probability under the model's base conditional distribution. As a result, additional independent sampling from the same distribution is unlikely to yield success.

Unlike prior scaling methods (Brown et al., 2024; Zuo and Zhu, 2026; Wang et al., 2025b),

which repeatedly sample from a fixed distribution $p(\cdot \mid x)$, our approach modifies the sampling distribution itself by conditioning on ICL demonstrations that draw from the model itself. However, in terms of the pure resource allocation aspect, we adopt an effective and strong adaptive allocation strategy that focuses compute on harder queries as easier ones are resolved. Building on this strong foundation, we further improve efficiency by adapting the sampling distribution via evolving ICL conditioning, rather than sampling repeatedly from a fixed distribution. We retain the strengths of adaptive resource allocation and further enhance it by adaptively shaping the generation distribution.

ICL construction via similarity-based selection. For each active query $x \in \mathcal{A}$, we construct an ICL prompt by selecting a set of demonstrations from D_{test}. We first embed each question using a fixed embedding function $\phi(\cdot)$ and measure semantic similarity using cosine similarity. The neighborhood of x is defined as the set of top-P most similar queries:

$$\mathcal{P}(x) = \text{Top-}P\Big(\mathcal{A} \setminus \{x\},\ \cos\big(\phi(x), \phi(\cdot)\big)\Big),$$

where $\cos(\cdot, \cdot)$ denotes the cosine similarity. For each neighbor question $z \in \mathcal{P}(x)$, let $\tilde{y}(z)$ denote the most recently generated response for z. In practice, since evaluation is deterministic and queries are eliminated once a correct response is found, $\tilde{y}(z)$ is typically the first correct response for z when available. The ICL prompt for x is then constructed as

$$\Pi(x) = \{(z, \tilde{y}(z)) \mid z \in \mathcal{P}(x)\}.$$

This prompt is used to condition subsequent generations for x.

ICL-conditioned sampling. Given the constructed prompt $\Pi(x)$, new responses are generated from the conditional distribution

$$y \sim p(\cdot \mid x, \Pi(x)).$$

These responses are still evaluated by the reward oracle, and the active set A is updated accordingly.

Algorithm 1 Self-Evolving Test-Time Allocation

Require: Test questions $\mathcal{S}$, language model p, reward oracle r, total generation rounds R, warm-up rounds R_{warm} ($R_{\text{warm}} \leq R$), per-round sample size K, elimination threshold γ, embedding function $\phi(\cdot)$, neighborhood size P.

Return: For each $x \in \mathcal{S}$, a set of generated samples $g(x)$ and a final answer $\tilde{y}(x)$.

Initialize response pool $g(x) \leftarrow \emptyset$ for all $x \in \mathcal{S}$ // stores all test-time generations
Initialize $\tilde{y}(x) \leftarrow \emptyset$ for all $x \in \mathcal{S}$
Initialize active set $\mathcal{A} \leftarrow \mathcal{S}$
Stage 1: Warm-up (Difficulty Estimation and $\mathcal{D}_{\text{test}}$ Construction) // fixed distribution sampling
for round $t = 1$ to R_{warm} **do**
  for each $x \in \mathcal{S}$ **do**
    Generate K responses $\{y_i\}_{i=1}^{K} \sim p(\cdot \mid x)$ // base distribution
    Append responses to $g(x)$
    if $\exists\, y \in g(x)$ such that $r(x, y) \geq \gamma$ **then**
      $\tilde{y}(x) \leftarrow y$ // first correct (deterministic verifier)
      $\mathcal{A} \leftarrow \mathcal{A} \setminus \{x\}$ // eliminate solved questions from the active set
Stage 2: Adaptive Distribution Sampling // self-improving ICL
Precompute question embeddings $\phi(x)$ for all $x \in \mathcal{S}$
for each $x \in \mathcal{S}$ **do**
  Define neighborhood:

$$\mathcal{P}(x) \leftarrow \text{Top-}P\big(\mathcal{S} \setminus \{x\},\ \cos(\phi(x), \phi(\cdot))\big)$$

// fixed neighbors; prompts evolve via $\tilde{y}(\cdot)$
for round $t = R_{\text{warm}} + 1$ to R **do**
  for each $x \in \mathcal{A}$ **do**
    Construct ICL prompt using most recent neighbor responses:

$$\Pi(x) \leftarrow \{(z, \tilde{y}(z)) : z \in \mathcal{P}(x),\ \tilde{y}(z) \text{ exists}\}$$

    // updates as $\mathcal{D}_{\text{test}}$ grows
    Generate K responses $\{y_i\}_{i=1}^{K} \sim p(\cdot \mid x, \Pi(x))$ // distribution shift via ICL
    Append responses to $g(x)$
    if $\exists\, y \in g(x)$ such that $r(x, y) \geq \gamma$ **then**
      $\tilde{y}(x) \leftarrow y$
      $\mathcal{A} \leftarrow \mathcal{A} \setminus \{x\}$ // eliminate upon success
return $\{g(x), \tilde{y}(x)\}_{x \in \mathcal{S}}$

5 Experiments

In this section, we present the evaluation of our approach among different datasets with different models.

5.1 Experimental settings

Datasets and models. We evaluate our method on four widely used reasoning and coding benchmarks: MATH-500 (Lightman et al., 2024), LiveCodeBench (Jain et al., 2025), MinervaMath (Lewkowycz et al., 2022), and GPQA-Diamond (Rein et al., 2024). For LiveCodeBench, we restrict our evaluation to questions released between

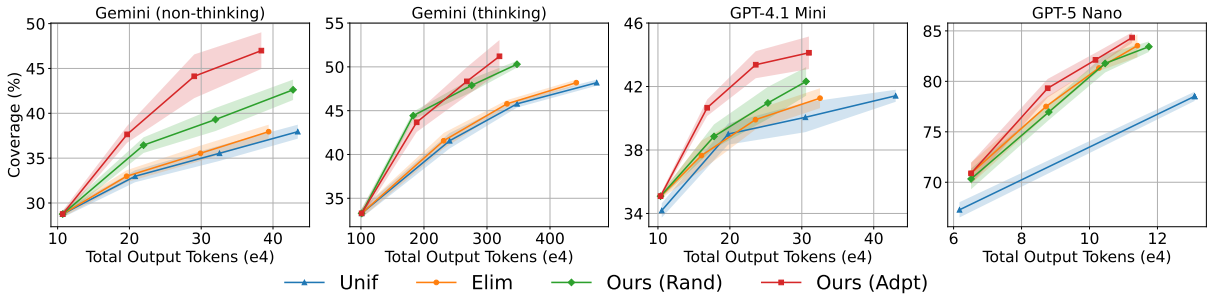


Figure 3: Coverage comparison on LiveCodeBench.

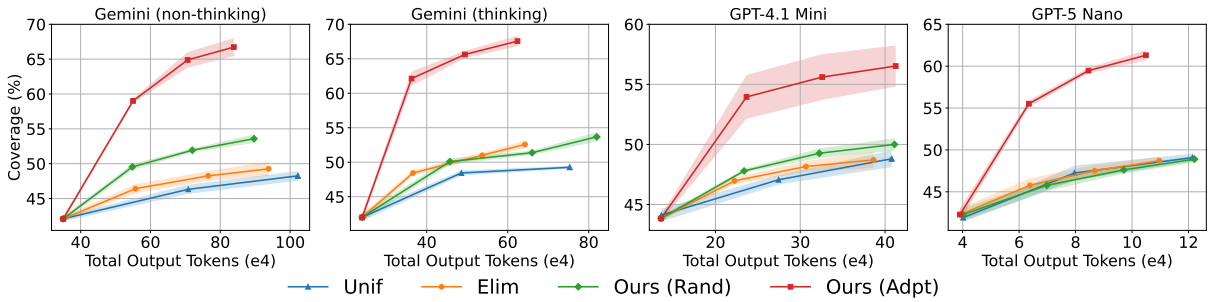


Figure 4: Coverage comparison on MinervaMath.

10/05/2024 and 01/04/2025, following the evaluation protocol reported for Gemini 2.5 (Team, 2025).

In addition to these benchmarks, we construct a custom evaluation dataset using Reasoning Gym (Stojanovski et al., 2025). This dataset consists of 400 math-oriented reasoning problems sampled at the *hard* difficulty level. Details of the dataset construction process are provided in Appendix A. For MATH-500 datasets, there are 500 math reasoning questions. There are 272 reasoning questions in the MinervaMath dataset, and 198 reasoning questions in GPQA-Diamond correspondingly. The selected LiveCodeBench dataset contains 166 test questions relating to coding.

We evaluate both reasoning and non-reasoning large language models drawn from recent frontier model families. Specifically, we use the `Gemini-2.5-flash-lite` model under two configurations: with internal thinking enabled and disabled. We also include `GPT-4.1 Mini`, a non-reasoning model, and `GPT-5 Nano`, a reasoning-oriented model. All models are accessed exclusively through their respective public APIs.

Baselines and metrics. We compare our method OURS (ADPT) against three methods. The first is the Best-of-N (UNIF) strategy (Snell et al., 2025), which uniformly allocates test-time compute by sampling N independent responses per query from a fixed conditional distribution and selecting the highest-scoring one using a reward oracle. The second method is the ELIMINATION (ELIM) method (Manvi et al., 2024; Zuo and Zhu, 2026), which formulates test-time scaling as an adaptive allocation problem. In this method, test questions that achieve a correct response early are eliminated from further sampling, while unresolved questions continue to receive additional samples. All samples for a given query are drawn from a fixed conditional distribution, and adaptation occurs solely through the allocation of the number of samples generated per question. The last method OURS (RAND) is a variant of our approach that uses *random* prompt selection. This method follows the same two-stage procedure as our method but replaces similarity-based demonstration selection with randomly chosen test-time examples.

All results are averaged over four random runs, with shaded regions in plots indicating ± 0.5 standard deviations. As stated in Algorithm 1, we adopt a round-based evaluation for all methods, where in each round, new responses are generated **only for test questions that remain unsolved**. By default, we use a total of $R = 4$ rounds and $R_{warm} = 1$. For the number of neighboring shots in adaptive rounds, we apply $P = 3$ for all our settings. For UNIF, ELIM, and the warm-up round, we apply zero-shot COT prompt (Kojima et al., 2022). We

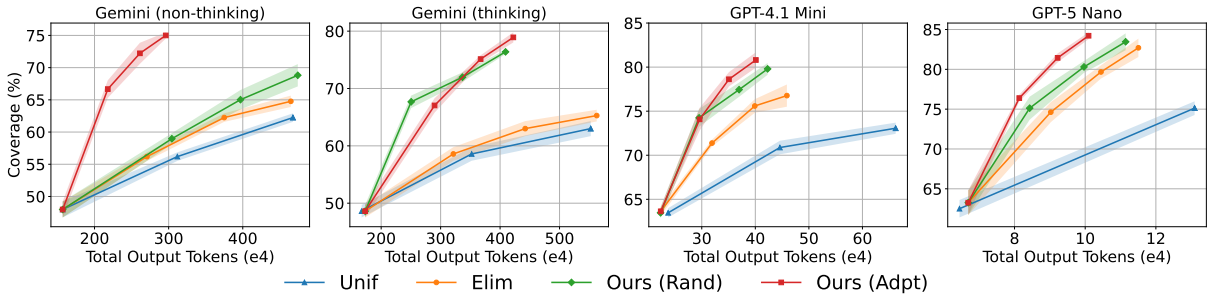


Figure 5: Coverage comparison on GPQA-Diamond.

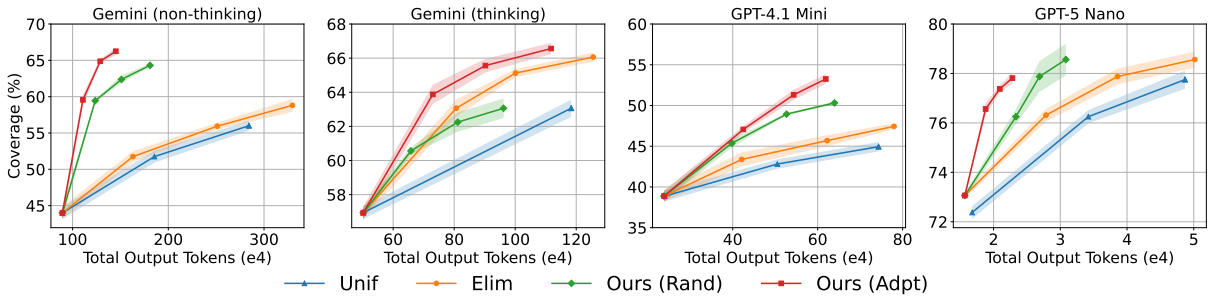


Figure 6: Coverage comparison on Reasoning Gym.

include all details in Appendix B. For UNIF, we match the same level of output token to that of ELIM to ensure a fair comparison. We report performance curves for all methods using the same random seeds. For ELIM, OURS (RAND), and OURS (ADPT), we use the same warm-up stage generation for efficiency purposes, which corresponds to the first point in each plot. For Gemini-family models, results are fully reproducible under identical seeds, and therefore, the first-point performance is exactly matched across methods. For GPT-based models, perfect seed replication is not supported, and as a result, the first-point performance may exhibit minor variation despite using the same seed.

As stated before, we primarily report *coverage* as our main evaluation metric because the *accuracy* is highly relevant to the *coverage* performance. Nevertheless, we provide the *accuracy* performance to complement our selection metrics in Section 5.3.

5.2 Overall experiment results

Benchmark results. Fig. 8 reports the final round *coverage* of all methods across all benchmarks and models. For the UNIF baseline, we propagate the current final-round performance to unfinished rounds (till R at 4) to enable a fair comparison under the same evaluation protocol.

Across all benchmarks and model families, our method consistently improves coverage relative to UNIF, demonstrating the benefit of adaptive test-time scaling over uniform sampling. Compared to ELIM, our approach yields additional gains on nearly all benchmarks, highlighting the advantage of distributional adaptation via evolving in-context demonstrations. The only exception occurs for `GPT-5 Nano` when compared against ELIM and OURS (RAND) on Reasoning Gym. This behavior is consistent with the relatively limited sensitivity of `GPT-5 Nano` to prompt-based conditioning on Reasoning Gym.

While random prompts can yield improvements in many settings, their performance is substantially less consistent. On reasoning-intensive benchmarks such as Reasoning Gym, MinervaMath, and LiveCodeBench, OURS (RAND) occasionally underperforms UNIF or ELIM, with the effect being more pronounced for reasoning-oriented models. In contrast, similarity-based demonstration selection consistently achieves larger and more stable gains across models and benchmarks, highlighting the importance of structured prompt construction for inducing beneficial distribution shifts during test-time scaling. Overall, these results demonstrate that our approach leads to robust and consistent coverage gains beyond some prior methods.

Output token efficiency. In addition to coverage, we report total output token usage alongside performance curves in Figs. 2 to 6. We measure output

token usage excluding the thinking summary for all reasoning models, as this summary is a simulated artifact and does not reflect the model's actual internal reasoning process for any of the evaluated models. Still, we include a representative result of total token usage in Appendix B. Across all benchmarks, our method consistently achieves the highest token efficiency among all compared approaches, for it requires fewer tokens to reach the same accuracy level. In particular, our approach typically exceeds the performance of ELIM, UNIF, and OURS (RAND) while consuming markedly fewer samples. These results demonstrate that our evolving distributional adaptation not only improves coverage but also yields strong gains in token efficiency across diverse tasks and models. We include an analysis of the reasons why we can achieve such token efficiency in Appendix C.

5.3 Ablation Studies

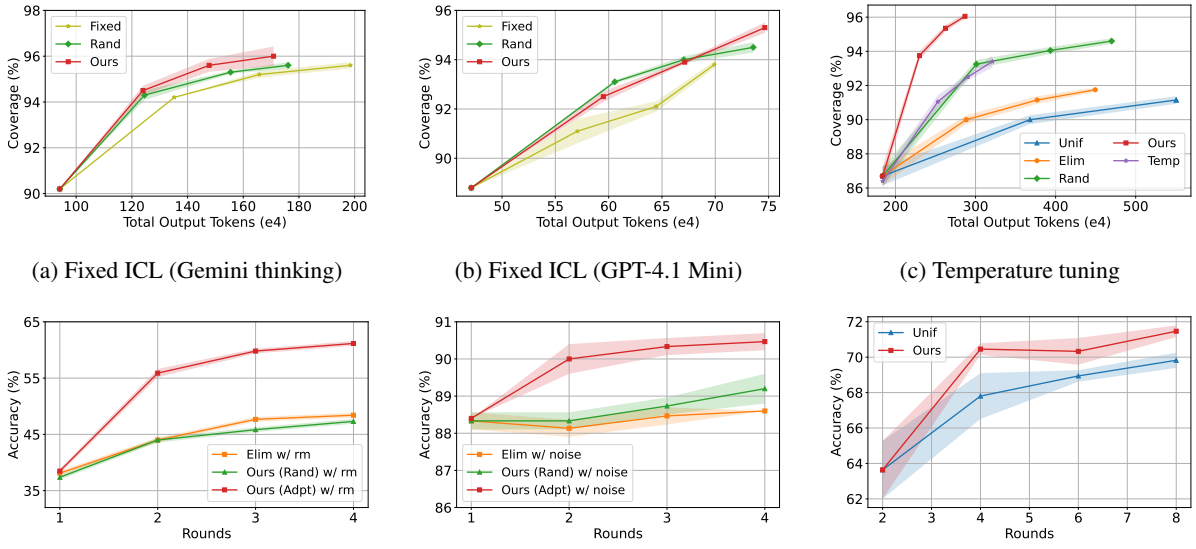
Fixed demonstrations. We further conduct an ablation study using a fixed set of in-context demonstrations with ELIM, while keeping all other settings identical to our proposed approach. Given the availability of demonstration prompts in prior work, we evaluate this fixed-prompt variant on the MATH-500 dataset using the `Gemini-2.5-flash-lite` model with thinking enabled and the `GPT-4 Mini` model. The demonstrations used in this experiment follow commonly adopted prompt templates from prior studies (Lewkowycz et al., 2022; Brown et al., 2024). The results are presented in Fig. 7. Our dynamic setting offers an advantage in both token efficiency and accuracy, even with random demonstration selection. Fixed demonstrations still lead to a static sampling distribution, for it does not introduce any adaptivity in the generation process.

Alternative sampling distribution. We have highlighted the importance of the sampling distribution in test-time scaling. Our framework is agnostic to the specific mechanism used to modify this distribution and can, in principle, accommodate any test-time intervention that reshapes model outputs. As a simple alternative to in-context learning, we consider straightforward temperature adjustment, which directly controls the entropy of the sampling distribution. We conduct an ablation study on the MATH-500 benchmark using the `Gemini-2.5-flash-lite` model with thinking disabled, where Algorithm 1 is modified to progressively increase the sampling temperature across rounds in place of updating ICL demonstrations. We start with a temperature $= 0.3$ and then increase by 0.4 every round. All other experimental settings are kept identical. The results, shown in Fig. 7 and marked as TEMP, illustrate the plug-and-play nature of our framework with respect to different sampling distribution controls.

Self-consistency. Other than the coverage metrics, we utilize self-consistency (Wang et al., 2023) as suggested in Wang et al. (2025a), which claims that self-consistency could have a good performance as a selection mechanism for reasoning models. We conduct experiments using `GPT-5 Nano` model on the GPQA-Diamond benchmark. We increase our rounds to $P = 8$ to build a better consensus pool, and we do not eliminate based on a ground-truth oracle. Instead, we eliminate once the response pool reaches a consensus over 75% as a variation for our Algorithm 1. Since self-consistency picks out a final answer, we report *accuracy* instead of *coverage*. To differentiate rounds from the previous plots, we mark rounds explicitly. Other settings are identical to our main experiments. We report our results in Fig. 7. Our method still outperforms the baselines with self-consistency selection. Also, given that our token efficiency has been shown sufficiently in other experiments, we do not explicitly show it, but our method has a better output token efficiency while achieving a better accuracy performance.

Reward-model. We further evaluate our method under a reward-model-based setting, where answer quality is assessed using a strong external model rather than direct ground-truth comparison. Empirically, we observe that Gemini-3-flash demonstrates stronger performance in the Math category compared to the current rewardbench (Lambert et al., 2025) leader (Gemini-2.5-pro) while also offering significantly lower latency. Therefore, we adopt Gemini-3-flash as our reward model for efficiency and competitiveness. We conduct this experiment using the Gemini-2.5-flash-lite model with thinking enabled on the MinervaMath dataset. The results in Fig. 7 are consistent with our earlier findings.

Additionally, to explicitly test robustness under imperfect verification, we introduce controlled noise into the reward model's outputs by randomly flipping binary correctness labels with a 5% probability. In this noisy-reward setting, the reward model determines whether a response is considered solved and whether it is added to the demonstration



(a) Fixed ICL (Gemini thinking) (b) Fixed ICL (GPT-4.1 Mini) (c) Temperature tuning

(d) Reward model (Gemini thinking) (e) Noisy reward (GPT-4.1 Mini) (f) Self-consistency

Figure 7: Ablation studies across fixed demonstrations, temperature tuning, and selection strategies.

pool; if no response is marked correct for a query, we fall back to selecting the most recent response. Ground-truth labels are used only for final evaluation and are not involved in adaptive allocation or demonstration construction. We conduct this experiment using GPT-4.1-mini on the MATH500 dataset. The results in Fig. 7 still remain consistent with our earlier findings.

6 Discussion

This work studies test-time scaling from a unified perspective that jointly considers how inference-time computation is allocated and how additional computation is used to influence the model's sampling behavior during inference. Rather than treating test-time scaling solely as generating more samples from a fixed process, we show that allowing the conditional generation behavior to evolve over time is a critical and complementary dimension. By leveraging evolving in-context demonstrations constructed from test-time generations, our framework enables the response process to adapt progressively, without access to external training data, model updates, or auxiliary allocation models. Empirically, this structured adaptation yields both improved coverage and higher token efficiency across diverse benchmarks and model families, consistently outperforming or matching strong adaptive baselines while consuming fewer inference tokens.

More broadly, our framework highlights the importance of temporal structure in test-time inference. As generations accumulate, intermediate outputs provide increasingly informative signals that can be reused to guide subsequent computation, enabling inference-time behavior to improve even under a fixed model. While we instantiate this idea through in-context learning, the framework itself is not limited to ICL and naturally accommodates alternative mechanisms for influencing generation, such as temperature control or other lightweight decoding strategies. As inference cost becomes an increasingly central concern, we believe adaptive test-time strategies that jointly allocate computation and allow generation behavior to evolve will play a key role in the practical and efficient deployment of large language models.

Limitations

Our framework relies on modifying inference-time behavior by conditioning generation on previously generated responses, which implicitly influences the model's effective sampling distribution. While this mechanism enables more efficient use of test-time compute, it also introduces sensitivity to the quality of intermediate generations. In particular, if early responses are noisy or poorly aligned with the target query, the resulting contextual conditioning may provide limited benefit or, in some cases, hinder subsequent generations. Although our warm-up phase and similarity-based selection mitigate this effect in practice, developing more robust strategies for selecting and filtering test-time demonstrations remains an important direction for future work.

In addition, our current instantiation focuses on in-context learning as a lightweight means of influencing generation behavior. While the proposed framework is general and can accommodate alternative mechanisms such as temperature adjustment or other decoding strategies, we do not explore these variants in this work. We view both aspects as a promising avenue for future work.

References

Emre Can Acikgoz, Cheng Qian, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. 2025. Self-improving llm agents at test-time. *Preprint*, arXiv:2510.07841.

Rishabh Agarwal, Avi Singh, Lei M Zhang, Bernd Bohnet, Luis Rosias, Stephanie C.Y. Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, John D Co-Reyes, Eric Chu, Feryal Behbahani, Aleksandra Faust, and Hugo Larochelle. 2024. Many-shot in-context learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Amanda Bertsch, Maor Ivgi, Emily Xiao, Uri Alon, Jonathan Berant, Matthew R. Gormley, and Graham Neubig. 2025. In-context learning with long-context models: An in-depth exploration. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 12119–12149, Albuquerque, New Mexico. Association for Computational Linguistics.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. 2009. Pure exploration in multi-armed bandits problems. In *International conference on Algorithmic learning theory*, pages 23–37. Springer.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sashank Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, and 48 others. 2023. Palm: scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24(1).

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. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. 2025. Learning how hard to think: Input-adaptive allocation of LM computation. In *The Thirteenth International Conference on Learning Representations*.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Babu Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. 2025. Context length alone hurts LLM performance despite perfect retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 23281–23298, Suzhou, China. Association for Computational Linguistics.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, and 3 others. 2022. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*.

Kevin Jamieson and Robert Nowak. 2014. Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting. In *2014 48th annual conference on information sciences and systems (CISS)*, pages 1–6. IEEE.

Adam Tauman Kalai, Ofir Nachum, Santosh S. Vempala, and Edwin Zhang. 2025. Why language models hallucinate. *Preprint*, arXiv:2509.04664.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *Preprint*, arXiv:2001.08361.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. 2025. RewardBench: Evaluating reward models for language modeling.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

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

Qingwen Lin, Boyan Xu, Guimin Hu, Zijian Li, Zhifeng Hao, Keli Zhang, and Ruichu Cai. 2025. Cmcts: A constrained monte carlo tree search framework for mathematical reasoning in large language model.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What makes good in-context examples for gpt-3? *Preprint*, arXiv:2101.06804.

Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. 2025. Can 1b LLM surpass 405b LLM? rethinking compute-optimal test-time scaling. In *Workshop on Reasoning and Planning for Large Language Models*.

Andrea Locatelli, Maurilio Gutzeit, and Alexandra Carpentier. 2016. An optimal algorithm for the thresholding bandit problem. *Preprint*, arXiv:1605.08671.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Rohin Manvi, Anikait Singh, and Stefano Ermon. 2024. Adaptive inference-time compute: Llms can predict if they can do better, even mid-generation. *Preprint*, arXiv:2410.02725.

Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. 2023. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 12284–12314, Toronto, Canada. Association for Computational Linguistics.

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

OpenAI. 2024. Learning to reason with llms. Accessed 12-Sep-2024.

Chengwei Qin, Aston Zhang, Chen Chen, Anirudh Dagar, and Wenming Ye. 2024. In-context learning with iterative demonstration selection. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7441–7455.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First conference on language modeling*.

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

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. 2025. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Hanshi Sun, Momin Haider, Ruiqi Zhang, Huitao Yang, Jiahao Qiu, Ming Yin, Mengdi Wang, Peter Bartlett, and Andrea Zanette. 2024. Fast best-of-n decoding via speculative rejection. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Zhendong Tan, Xingjun Zhang, Chaoyi Hu, Yancheng Pan, and Shaoxun Wang. 2025. Adaptive rectification sampling for test-time compute scaling. *Preprint*, arXiv:2504.01317.

Eshaan Tanwar, Subhabrata Dutta, Manish Borthakur, and Tanmoy Chakraborty. 2023. Multilingual LLMs are better cross-lingual in-context learners with alignment. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6292–6307, Toronto, Canada. Association for Computational Linguistics.

Google Gemini Team. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *Preprint*, arXiv:2507.06261.

Jens Tuyls, Dylan J Foster, Akshay Krishnamurthy, and Jordan T. Ash. 2026. Representation-based exploration for language models: From test-time to post-training. In *The Fourteenth International Conference on Learning Representations*.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*.

Junlin Wang, Shang Zhu, Jon Saad-Falcon, Ben Athiwaratkun, Qingyang Wu, Jue Wang, Shuaiwen Leon Song, Ce Zhang, Bhuwan Dhingra, and James Zou. 2025a. Think deep, think fast: Investigating efficiency of verifier-free inference-time-scaling methods. *Preprint*, arXiv:2504.14047.

Xinglin Wang, Shaoxiong Feng, Yiwei Li, Peiwen Yuan, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. 2025b. Make every penny count: Difficulty-adaptive self-consistency for cost-efficient reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6919–6932, Albuquerque, New Mexico. Association for Computational Linguistics.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Yuheng Wu, Azalia Mirhoseini, and Thierry Tambe. 2025. On the role of temperature sampling in test-time scaling. *Preprint*, arXiv:2510.02611.

Fanzeng Xia, Yidong Luo, Tinko Sebastian Bartels, Yaqi Xu, and Tongxin Li. 2025. Rethinking the unsolvable: When in-context search meets test-time scaling. *Preprint*, arXiv:2505.22290.

Benfeng Xu, Quan Wang, Zhendong Mao, Yajuan Lyu, Qiaoqiao She, and Yongdong Zhang. 2023. kNN prompting: Beyond-context learning with calibration-free nearest neighbor inference. In *The Eleventh International Conference on Learning Representations*.

Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. 2025. Hallucination is inevitable: An innate limitation of large language models. *Preprint*, arXiv:2401.11817.

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

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. 2024. Making retrieval-augmented language models robust to irrelevant context. In *The Twelfth International Conference on Learning Representations*.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 10495–10516, Vienna, Austria. Association for Computational Linguistics.

Yinglun Zhu, Sumeet Katariya, and Robert Nowak. 2020. Robust outlier arm identification. In *International Conference on Machine Learning*, pages 11566–11575. PMLR.

Yinglun Zhu, Julian Katz-Samuels, and Robert Nowak. 2022. Near instance optimal model selection for pure exploration linear bandits. In *International Conference on Artificial Intelligence and Statistics*, pages 6735–6769. PMLR.

Yinglun Zhu, Dongruo Zhou, Ruoxi Jiang, Quanquan Gu, Rebecca Willett, and Robert Nowak. 2021. Pure exploration in kernel and neural bandits. *Advances in neural information processing systems*, 34:11618–11630.

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

A Other details for experiments

Reasoning-Gym Dataset Construction. To evaluate our method on diverse forms of symbolic and algorithmic reasoning, we construct a balanced benchmark using the Reasoning Gym (Stojanovski et al., 2025) framework. We focus on four representative categories that capture complementary reasoning skills: *decimal arithmetic*, *advanced geometry*, *binary alternation*, and *letter counting*. For each category, we sample 100 problem instances using a fixed random seed to ensure reproducibility, resulting in a total of 400 evaluation problems.

For each category, we instantiate the corresponding Reasoning Gym task using its canonical dataset interface and verify correctness using the task-specific scoring function provided by the framework. All generated instances include the problem statement, ground-truth answer, and metadata identifying the originating task. We aggregate the resulting problems into a unified evaluation set, which is used consistently across all methods and experimental settings.

Model APIs and decoding settings. All experiments are conducted using publicly available LLM APIs. Unless otherwise specified, response generation uses a temperature of 0.3 when temperature control is supported. We set `Top_P` $= 0.9$ and `Top_K` $= 40$ for all models that expose these parameters.

For `Gemini` family models, we use the maximum available output token limit of $65{,}536$ tokens. When *thinking* is enabled, we allocate a maximum thinking budget of $24{,}576$ tokens. For all `GPT` family models, we set the maximum output length to $32{,}768$ tokens. The reasoning and verbosity levels are all medium. To get the question embeddings, we mainly use Gemini's text embedding model.

Prompt construction for MATH-500. We used the same prompt across different models. Our method uses two prompt variants: a *warm-up prompt* used for initial sampling, and an *adaptive prompt* that prepends P semantically similar test-time demonstrations (neighbors) before querying the target problem. Illustrative examples used in our MATH-500 experiment are shown below.

The warm-up stage uses a zero-shot CoT (Kojima et al., 2022) prompt shared across all models.

MATH-500 warm-up prompt

Reason step by step and put the final answer in \boxed{}.
Problem:
{*Problem statement*}

Solution:

We do not apply any complicated prompting engineering at this stage. We simply feed in the previously generated D_{test} demonstrations into the prompt constructions. Each example consists of a previously generated question–solution pair. The target question is appended after a separator.

MATH-500 adaptive prompt

Example 1
Problem:
{*Neighbor problem 1*}

Solution:
{*Neighbor solution 1*}

Example 2
Problem:
{*Neighbor problem 2*}

Solution:
{*Neighbor solution 2*}

Example 3
Problem:
{*Neighbor problem 3*}

Solution:
{*Neighbor solution 3*}

Now solve the following. Put the final answer in \boxed{}.
Problem:
{*Target question*}

Solution:

Prompt construction for LiveCodeBench. For LiveCodeBench, we use a code-only generation protocol: the model must output *only* valid Python 3 source code (no explanations, no comments, and no markdown fences). We employ two prompt variants: (i) a *base prompt* used in warm-up sampling and baseline settings, and (ii) an *adaptive neighbor+base prompt* that prepends up to three previously solved test-time examples before appending the base prompt verbatim.

The base prompt provides the problem statement, optional starter code, and explicit I/O and environment constraints.

LiveCodeBench warm-up prompt

You are a competitive programming assistant. Return ONLY valid Python 3 code that solves the task. Do not include explanations, comments, or markdown fences.

Problem:
{*question_content*}

Starter (optional):
{*starter_code or N/A*}

I/O requirements:

- Read from STDIN exactly as described in the problem.
- Write only the required answer(s) to STDOUT (no extra prints).
- Do not read/write files. Do not use network.
- Avoid heavy/obscure libraries.

Constraints:

- Python 3.10+.
- Prefer iterative solutions if deep recursion could occur.
- If multiple test cases exist, handle them all.

Return ONLY the Python source code.

In the adaptive stage, we prepend previously solved coding problems and their Python solutions (selected from test-time generations), followed by the base prompt verbatim. This construction provides retrieval-style guidance while preserving the original task specification.

LiveCodeBench adaptive prompt

You will see up to 3 previously solved coding problems with their Python 3 solutions.
Use them as guidance. After the examples, follow the final instructions below.
Return ONLY valid Python 3 code (no explanations, comments, or markdown fences).

Example 1
Problem:
{*neighbor_problem_1*}

Solution (Python 3 code only):
{*neighbor_solution_1*}

Example 2
Problem:
{*neighbor_problem_2*}

Solution (Python 3 code only):
{*neighbor_solution_2*}

Example 3
Problem:
{*neighbor_problem_3*}

Solution (Python 3 code only):
{*neighbor_solution_3*}

Base prompt (appended verbatim):
{*LiveCodeBench Base Prompt content*}

MinervaMath adaptive prompt

Example 1
Problem:
{*Neighbor problem 1*}

Solution:
{*Neighbor solution 1*}

Example 2
Problem:
{*Neighbor problem 2*}

Solution:
{*Neighbor solution 2*}

Example 3
Problem:
{*Neighbor problem 3*}

Solution:
{*Neighbor solution 3*}

_

Let's solve this step by step and put the final answer in \boxed{} with just the numeric value.
Problem:
{*Target problem*}

Solution:

Prompt construction for MinervaMath. For MinervaMath, the setting is similar to that of the MATH-500's.

Prompt construction for GPQA-Diamond. For GPQA, we use a multiple-choice solving protocol that asks the model to reason step by step and output exactly one option letter in boxed{} (A/B/C/D).

MinervaMath warm-up prompt

Let's solve this step by step and put the final answer in \boxed{} with just the numeric value.
Problem:
{*Problem statement*}

Solution:

GPQA warm-up MCQ prompt

Solve the problem step by step.
Select one selection and answer A, B, C or D in \boxed{}.

Question: {*question_text*}

GPQA adaptive prompt

You will see up to 3 previously answered multiple-choice questions.
Use them as guidance. After the examples, solve the question step by step.
Select one selection and answer A, B, C or D in \boxed{}.

–

Example 1
{*neighbor_question_1*}
Answer: {*neighbor_answer_1*}

–

Example 2
{*neighbor_question_2*}
Answer: {*neighbor_answer_2*}

–

Example 3
{*neighbor_question_3*}
Answer: {*neighbor_answer_3*}

–

Solve the problem step by step.
Select one selection and answer A, B, C or D in \boxed{}.

Question: {*target_question_text*}

Prompt construction for Reasoning Gym. For Reasoning Gym, we use a minimal answer-format instruction that requires the final answer to be enclosed in boxed{}. In contrast to math benchmarks, where explicit zero-shot CoT prompting is commonly used, we find it not useful in this setting. Thus, we apply a prompt without it.

Reasoning Gym warm-up prompt

Put the final answer within \boxed{}.
Problem:
{*Problem statement*}

Solution:

Reasoning Gym adaptive prompt

Example 1
Problem:
{*Neighbor problem 1*}

Solution:
{*Neighbor solution 1*}

Example 2
Problem:
{*Neighbor problem 2*}

Solution:
{*Neighbor solution 2*}

Example 3
Problem:
{*Neighbor problem 3*}

Solution:
{*Neighbor solution 3*}

–

Now solve the following. Put the final answer within \boxed{}.
Problem:
{*Target question*}

Solution:

B Additional experimental results

Additional allocation method. We incorporate an additional adaptive baseline inspired by (Wang et al., 2025b), referred to as Difficulty-Aware allocation DA. In this implementation, we use Gemini-3-flash to estimate per-question difficulty and allocate computation proportionally. Unlike the original batch-based formulation in (Wang et al., 2025b), we perform per-query difficulty estimation, which allows a more fine-grained allocation strategy aligned with our test-time adaptive setting.

To ensure a fair comparison, we strictly control the computational budget. Specifically, we maintain approximately the same total token usage as the Elim baseline, so that any performance difference reflects allocation strategy rather than increased computation. Given that token usage is matched and fully reported, we present results primarily across rounds to highlight differences in allocation dynamics. We can see that our approach still outperforms all other variations.

Method	R0	R1	R2	R3
DA	41.5	48.2	51.1	51.5
Elim	41.5	47.8	51.8	52.9
Ours (Rand)	41.5	48.9	50.7	52.2
Ours (Adpt)	41.5	60.3	64.3	66.2

Table 2: Coverage (%) across rounds on MinervaMath using Gemini-2.5-flash-lite (thinking).

Method	R0	R1	R2	R3
DA	88.8	90.6	91.8	92.2
Elim	88.8	91.0	92.4	92.4
Ours (Rand)	88.8	91.2	93.2	94.4
Ours (Adpt)	88.8	92.8	93.8	95.6

Table 3: Coverage (%) across rounds on MATH500 using GPT-4.1-mini.

Token count for all usage. We select GPQA-Diamond, using `Gemini-2.5-flash-lite` model with thinking enabled, as a representative benchmark for token-usage analysis, since it has the longest average output token length among all evaluated datasets. The setting is identical to our main experiments. The results, reported in Table 4, show that our method significantly reduces token consumption at every round. Importantly, because our approach also yields higher coverage, the true efficiency gains are underestimated by token counts alone.

Method	Round 0	Round 1	Round 2	Round 3	Total
ELIM	2,025,345	1,680,173	1,308,144	1,249,419	6,263,081
OURS	2,025,345	1,102,486	902,109	741,611	4,771,551
Savings (Elim–Ours)	0	577,687	406,035	507,808	1,491,530

Table 4: Average token usage (input prompt + thinking summary + response token) per round across 4 seeds.

Numerical results. We present numerical results for the approach OURS (ADPT) compared with all the other variants. We compare the last round coverage gain for OURS (ADPT) over the other baselines. In general, our method is able to achieve a good performance gain.

Benchmark	Nonthink	Think
MATH-500	4.28/4.08/1.08	1.82/1.07/0.02
LiveCodeBench	7.43/7.43/3.46	2.56/2.56/0.20
MinervaMath	15.93/15.36/11.86	16.11/14.43/13.39
GPQA	11.07/10.23/7.03	12.16/11.40/1.68
ReasonGym	8.98/8.06/1.52	2.27/0.58/3.38

Table 5: Coverage improvement (%) for Gemini models (U/E/R).

Benchmark	GPT-4.1 Mini	GPT-5 Nano
MATH-500	2.35/1.08/0.22	1.68/1.35/0.33
LiveCodeBench	2.55/3.11/2.01	3.41/1.14/1.67
MinervaMath	7.14/7.41/6.34	10.29/11.43/11.34
GPQA	5.50/3.25/0.67	5.56/1.68/1.05
ReasonGym	6.31/5.04/2.33	0.31/-0.02/0.31

Table 6: Coverage improvement (%) for GPT models (U/E/R).

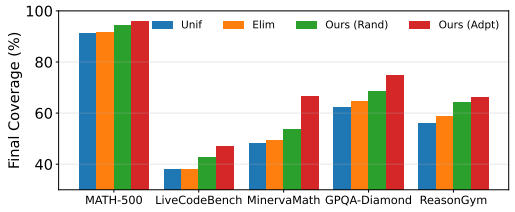
C Token efficiency analysis

We attribute the token efficiency of our approach to two complementary factors: (i) the characteristics of modern long-context LLMs, and (ii) the ability to resolve queries that are effectively unsolvable under fixed distribution sampling.

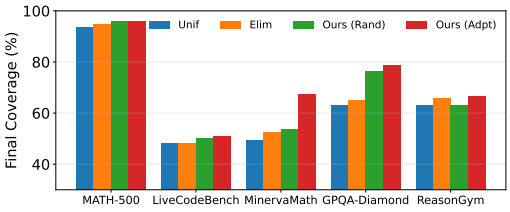
Long context windows and token amplification. Modern LLMs—especially reasoning models (DeepSeek-AI et al., 2025; Team, 2025)—operate with increasingly long context windows. The explicit reasoning or "thinking" process alone can consume thousands of tokens, and generating a complete final response often requires additional substantial output. As a result, a single failed attempt on a hard query can already incur a large token cost. This issue is further exacerbated for difficult or ambiguous queries, where models may hallucinate extended reasoning chains without converging to a formatted answer (Xu et al., 2025; Kalai et al., 2025). The same applies to powerful non-reasoning models, which often have large output budgets and can exhibit similar behavior when repeatedly sampling on challenging inputs.

Under a static sampling setting, hard queries tend to dominate the overall token budget: repeatedly allocating samples to such queries often leads to multiple long, unsuccessful generations, significantly inflating total token usage. Consequently, naively increasing the number of samples does not necessarily improve efficiency and can instead amplify token consumption.

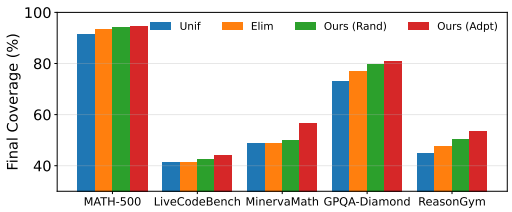
Solving unsolvable questions. The second—and more critical—factor behind token efficiency is the ability to actually resolve questions that are unlikely to be solved under static sampling. Recent work by Xia et al. (2025) shows that ICL can unlock solution paths for queries that appear unsolvable when sampled without ICL demonstrations. By conditioning inference on carefully selected demonstrations, ICL can shift the model toward



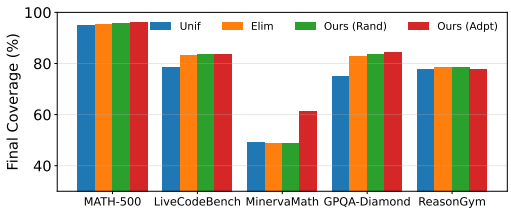
(a) Gemini (non-thinking)



(b) Gemini (thinking)



(c) GPT-4 Mini



(d) GPT-5 Nano

Figure 8: Final round coverage comparison

more effective reasoning trajectories, particularly for complex problems. Our approach leverages this insight by incorporating ICL into our self-improving scaling scheme. Once a hard question is successfully solved, further expensive generations for that question are avoided, leading to substantial savings in output tokens. Importantly, the benefit is multiplicative: resolving a previously unsolvable question not only prevents repeated long hallucinated outputs for that question, but can also provide informative demonstrations that help guide inference on related queries. Overall, token efficiency in our framework arises not merely from allocating fewer samples but from increasing the probability of early success on difficult queries. By combining adaptive allocation with ICL-driven distributional shifts, our method reduces redundant long generations and makes more effective use of the large context windows available in modern LLMs.

D Failure mode

To assess the robustness of similarity-based evolving ICL under weak semantic structure, we construct a deliberately heterogeneous test set. Specifically, we combine 60 questions drawn from LiveCodeBench, MATH500, and GPQA-Diamond (20 each), and further enforce diversity by selecting semantically distant instances based on cosine similarity in embedding space. This procedure intentionally disrupts local clustering and weakens nearest-neighbor relationships. The results are shown in Table 7. While the absolute gains are smaller than those observed on more structured benchmarks, our method continues to yield consistent improvements over baseline strategies without any performance degradation. For example, at Round 3, OURS (ADPT) achieves 55% coverage compared to 53.3% for ELIM. These results indicate that the benefits of evolving ICL are more limited in highly heterogeneous settings, though it still provides modest improvements over baseline approaches.

Method	R0	R1	R2	R3
Elim	45.0	53.3	53.3	53.3
Ours (Rand)	45.0	51.6	55.0	55.0
Ours (Adpt)	45.0	53.3	55.0	55.0

Table 7: Coverage (%) on a deliberately diverse dataset.

E The use of Large Language Models

LLMs were used to polish the writing of this paper.