SeedThink: Adaptive Seed-Thought Initialization for Efficient Test-Time Reasoning

Anonymous ACL submission

Abstract

Large reasoning models (LRMs) achieve strong performance through extended chains of thought, but their deliberative reasoning often incurs substantial inference overhead. We propose **SeedThink**, an adaptive seed-initialization framework that reduces unproductive overthinking by warm-starting deliberative reasoning with concise seed thoughts. Our key observation is that seed thoughts can guide LRMs toward more efficient reasoning trajectories, while the optimal seed granularity varies across problems, from high-level outlines to detailed solution sketches. SeedThink therefore learns a shared embedding space and selects seed-generation strategies via nearest-prototype similarity matching. It further accelerates inference with **self-enhanced speculative decoding**, which reuses the selected seed as a model-free draft for the subsequent seed-conditioned reasoning stage. Experiments on mathematical, financial, and scientific reasoning benchmarks show that SeedThink substantially reduces inference cost while preserving competitive accuracy, achieving up to a **4.1×** **end-to-end speedup** and a **68% reduction in generation length** with minimal accuracy degradation.

1 Introduction

Large language models (LLMs) increasingly benefit from test-time compute, a phenomenon commonly referred to as test-time scaling (Snell et al., 2024; Muennighoff et al., 2025). Recent large reasoning models (LRMs), such as OpenAI o1 (Jaech et al., 2024), DeepSeek-R1 (Guo et al., 2025a), and Qwen3 (Yang et al., 2025), improve performance by generating extended chains of thought with reflection, backtracking, and self-verification (Wei et al., 2022). This deliberative *Thinking* mode is especially effective on complex mathematical and scientific reasoning benchmarks (Cobbe et al., 2021; Hendrycks et al., 2021; He et al., 2024; Rein et al., 2024; Lewkowycz et al., 2022), but it also introduces substantial inference cost.

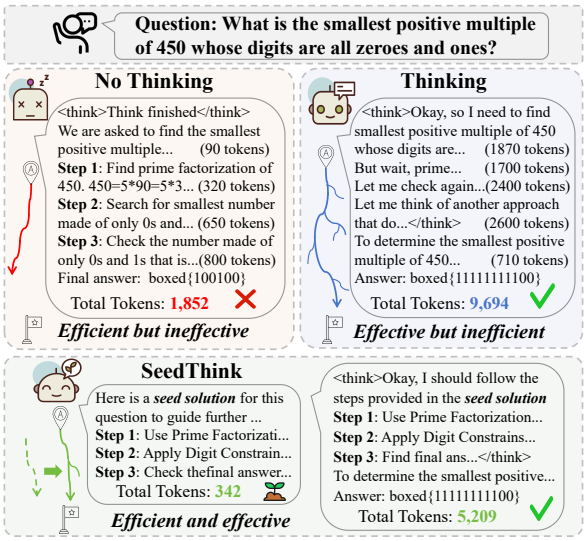


Figure 1: Comparison of *No-Thinking*, *Thinking*, and SeedThink on a MATH example (Hendrycks et al., 2021). SeedThink initializes reasoning with a concise seed that guides exploration toward productive regions and reduces unnecessary branches.

A major source of this cost is *overthinking*: LRMs often spend many tokens on repetitive verification, unproductive branches, and unnecessary exploration even after a sufficient solution path has been found (Chen et al., 2024; Cuadron et al., 2025). Existing mode-switching methods address this issue by choosing between efficient *No-Thinking* and more reliable *Thinking* modes according to input difficulty (Zhang et al., 2025a,b; Li et al., 2025a; Luo et al., 2025; Liang et al., 2025; Ha et al., 2025; Xing et al.). However, switching modes does not address the main bottleneck on challenging inputs, where most computation is still spent inside the *Thinking* trajectory itself. Moreover, reasoning is highly path-dependent: early steps strongly influence which solution regions the model explores, and the same token budget can

lead to different outcomes depending on how reasoning is initialized (Fan et al., 2025; Lin et al., 2025; Wu et al., 2025b).

This paper explores a complementary direction: **seed-initialized test-time reasoning**, where a concise seed thought is used to warm-start and guide the subsequent reasoning trajectory. Instead of asking the LRM to reason from scratch, we first generate a concise seed in the efficient *No-Thinking* mode and then condition the *Thinking* trajectory on this seed. As illustrated in Figure 1, *No-Thinking* is fast but often myopic, whereas standard *Thinking* explores more broadly but incurs large token overhead. SeedThink uses the seed to provide pivot reasoning anchors, such as subgoals, key lemmas, variable definitions, or candidate equation forms, guiding the model toward productive regions while pruning unnecessary exploration.

A key question is whether a single seed format is sufficient. Our pilot analysis suggests that seed granularity directly controls the trade-off between guidance and flexibility. Detailed seeds can collapse the search space and improve efficiency on easier problems, while higher-level seeds preserve flexibility on harder problems where overly specific guidance may be misleading. Motivated by this observation, SeedThink does not rely on a fixed seed template. Instead, it formulates seed selection as a utility-driven decision problem and trains a lightweight selector to route each problem to an appropriate seed-generation strategy via prototype matching in a shared embedding space.

Beyond guiding reasoning, seed-thoughts also enable an additional acceleration mechanism. Since the second-stage reasoning trajectory is explicitly conditioned on the first-stage seed, the two stages exhibit strong token-level alignment. We exploit this property through **Self-enhanced Speculative Decoding (SSD)**, which reuses seed tokens as a model-free draft for the subsequent reasoning stage. We further provide a token-alignment guarantee that lower-bounds the probability of seed n-gram reuse, explaining why seed-conditioned reasoning can yield high speculative acceptance rates (Leviathan et al., 2023; Li et al., 2024; Cai et al., 2024; Gloeckle et al., 2024; Li et al., 2025b; Sun et al., 2025).

Our contributions are summarized as follows.

- We present a systematic study showing that seed granularity has a structured impact on LRM accuracy and efficiency, and that no single seed strategy is universally optimal.
- We formulate seed selection as a utility-driven routing problem and introduce a lightweight prototype-based selector that matches each problem to an appropriate seed-generation strategy.
- We propose SeedThink, a unified framework that couples *No-Thinking* seed generation with seed-conditioned *Thinking*, and further reuses the seed for self-enhanced speculative decoding. Extensive experiments show substantial token and runtime reductions while maintaining competitive accuracy.

2 Related Work

Efficient reasoning. Recent work has explored various ways to reduce the cost of test-time reasoning in large reasoning models. One line of work adaptively switches between fast and slow reasoning modes using external classifiers or the model's own task-complexity estimates (Zhang et al., 2025c; Xing et al.; Zhang et al., 2025a; Liang et al., 2025). Another line reduces unnecessary reasoning by self-evaluation, early stopping, compact reasoning supervision, or length-aware training objectives (Wang et al., 2025c; Chen et al., 2024; Hou et al., 2025; Xia et al., 2025; Chen et al., 2025; Fang et al., 2025). Multi-model methods further use auxiliary evaluators, reward models, or smaller thought proposers to guide or accelerate reasoning (Fu et al., 2024; Kuhn et al., 2023; Sun et al., 2024; Pan et al., 2025; Wang et al., 2025b,d). Unlike mode-switching methods that choose between *Thinking* and *No-Thinking*, SeedThink couples the two modes: it first uses *No-Thinking* to generate a concise seed and then uses the seed to initialize and steer *Thinking*.

Planning-based prompting. Planning-based prompting methods use structured anchors to guide generation. Plan-and-Solve (Wang et al., 2023) decomposes problems into subtasks, Skeleton-of-Thought (Ning et al., 2024) expands a high-level skeleton in parallel, and CoThink (Fan et al., 2025) uses an instruction model to draft an outline before reasoning. These methods show that anchors are useful, but they largely use a fixed anchor format across problems. SeedThink instead treats the anchor as an adaptive reasoning initialization: it studies how seed granularity affects the trade-off between accuracy and efficiency, learns to select the appropriate seed strategy per input, and

further reuses the selected seed for speculative acceleration.

Speculative decoding. Speculative decoding accelerates inference through a draft-verify pipeline, where a lightweight drafter proposes candidate tokens and the target model verifies them in parallel (Leviathan et al., 2023; Chen et al., 2023). Subsequent work improves the drafter using n-gram retrieval, auxiliary models, or system-level optimizations (He et al., 2023; Zhao et al., 2024; Luo et al., 2024; Hu et al., 2024; Oliaro et al., 2025; Li et al., 2024, 2025b; Cai et al., 2024; Liu et al., 2024; Gloeckle et al., 2024; Miao et al., 2024). Our speculative acceleration is complementary: instead of training a separate drafter, SeedThink reuses the generated seed as a self-compatible draft for the subsequent seed-conditioned reasoning stage, exploiting the natural token-level alignment between the two stages.

3 Preliminaries and Observations

To establish our setting, we first fix notation and delineate the generation paradigms considered in this work. Subsequently, we investigate how the level of details in seed thoughts affects the reasoning trajectories of LRMs.

3.1 Preliminaries

Let M denote a standard LRM. Given an input question q, the model generates a response sequence r. Two special system tokens, $\langle \mathcal{T}_{begin} \rangle$ and $\langle \mathcal{T}_{end} \rangle$, are used to demarcate the beginning and end of the explicit reasoning block.

Thinking Mode. The model generates a full chain-of-thought reasoning process before producing the final answer. The generation can be formally described as:

$$\begin{aligned} r_{\text{thinking}} &= M(q + \langle \mathcal{T}_{begin} \rangle) \\ &= \langle \mathcal{T}_{begin} \rangle [\text{Thoughts}] \langle \mathcal{T}_{end} \rangle [\text{Conclusion}] \end{aligned}$$

No-Thinking Mode. The model is prompted to bypass the explicit reasoning process and generate the final answer directly. This is achieved by prepending a specific instruction inside the reasoning block that forces an immediate conclusion. The generation is:

$$\begin{aligned} r_{\text{nothinking}} &= M(q + \langle \mathcal{T}_{begin} \rangle + s_{\text{skip}} + \langle \mathcal{T}_{end} \rangle) \\ &= [\text{Conclusion}] \end{aligned}$$

where s_{skip} is a fixed string (e.g., "`Okay, I think I have finished thinking.`") that signals the model to skip deliberative thinking (Ma et al., 2025). $\langle \mathcal{T}_{begin} \rangle$ and $\langle \mathcal{T}_{end} \rangle$ are normally `<think>` and `</think>`, respectively

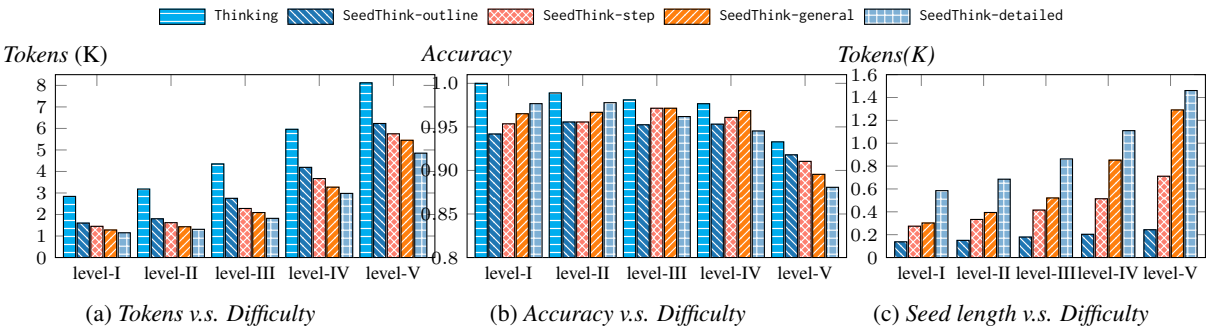
SeedThink Mode. *SeedThink* performs response generation in two stages: a seed generation stage and a reasoning stage. Given q, a seed s_{seed} is first generated, after which the model produces the final response conditioned on both the input question q and the seed s_{seed}:

$$\begin{aligned} r_{\text{SeedThink}} &= M(q + s_{seed} + \langle \mathcal{T}_{begin} \rangle) \\ &= \langle \mathcal{T}_{begin} \rangle [\text{Thoughts}] \langle \mathcal{T}_{end} \rangle [\text{Conclusion}] \end{aligned}$$

3.2 Observations

We conduct pilot experiments on MATH500 (Hendrycks et al., 2021) to examine how the *granularity* of seed thoughts affects downstream reasoning. We consider four granularity-controlled seed strategies: *outline*, which provides high-level strategic guidance; *key-steps*, which lists the main intermediate steps; *general-approach*, which gives a brief step-by-step plan; and *detailed-steps*, which provides a more concrete partial solution. For each problem, we first generate a seed using the *No-Thinking* mode with one of these strategies, and then run a second-pass *Thinking* generation conditioned on both the question and the seed. Figure 2 reports the main results using Qwen3-8B; additional model results are provided in Appendix C, and full prompt templates are provided in Appendix K.

Results and analysis. Figure 2 shows that no single seed strategy is universally optimal. Detailed seeds are more effective on easier problems, where concrete guidance reduces unnecessary exploration; balanced seeds work well on medium-difficulty problems; and coarse outline seeds are more robust on the hardest problems, where overly specific guidance may cause premature commitment. This pattern reveals a guidance–flexibility trade-off: stronger seeds can accelerate reasoning when the solution path is clear, while coarser seeds preserve flexibility under uncertainty. Since the best granularity varies across difficulty levels, a fixed seeding policy is inherently limited. This motivates our problem-conditioned selector, which learns to match each input with a suitable seed-generation strategy in Section 4.



(a) *Tokens v.s. Difficulty* (b) *Accuracy v.s. Difficulty* (c) *Seed length v.s. Difficulty*

Figure 2: Comparison of *Thinking* and four *SeedThink* variants on Qwen3-8B over different difficulty levels of the MATH500 dataset. Difficulty levels (I–V) are *dataset-provided labels* and increase monotonically from level-I (easiest) to level-V (hardest). (a) Average token usage versus difficulty; the reported token counts already include the tokens consumed by seed generation. (b) Accuracy versus difficulty for different seed granularities. (c) Average seed length across difficulty levels. The figure illustrates systematic differences in efficiency and accuracy induced by seed granularity.

4 Method

As illustrated in Figure 3, we first formalize the seed selection objective in Section 4.1. Section 4.2 then learns a utility-supervised embedding space that routes each problem to an appropriate seed-generation method (Figure 3a). Section 4.3 describes the two-stage generation pipeline, where a seed-thought is generated in *No-Thinking* mode and then used to condition the final *Thinking* trace (Figure 3b). Finally, Section 4.4 shows how the seed is reused as a draft for speculative decoding enhancement, leveraging token-level alignment for additional acceleration (Figure 3c).

4.1 Problem Formulation

SeedThink aims to *adaptively* choose a seed-generation strategy for each problem to balance solution quality and generation cost. Given an input problem q, our framework proceeds in two stages: (i) select a seed-thought generator from a finite candidate set, and (ii) generate the final reasoning trajectory conditioned on the selected seed-thought. The central challenge is thus to learn a selector that maps q to its utility-maximizing seed method.

Let $\mathcal{G} = \{g_1, \ldots, g_m\}$ denote the set of available seed-generation methods. For each method $g \in \mathcal{G}$ and problem q, we evaluate two quantities: final-answer correctness and inference cost.

Correctness. We use a binary correctness signal

$$A(g, q) \in \{0, 1\}, \tag{1}$$

where $A(g, q) = 1$ if the final answer produced using seed method g is correct, and $A(g, q) = 0$ otherwise.

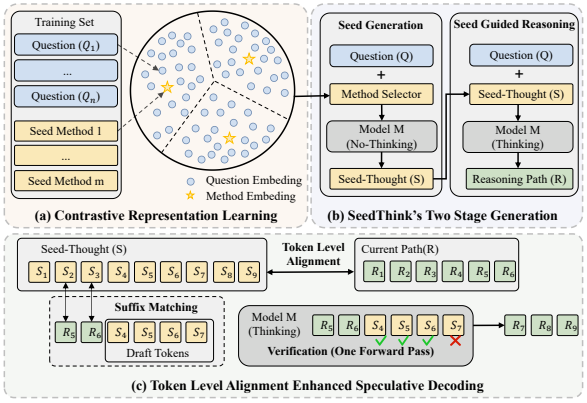


Figure 3: Overview of the SeedThink framework. (a) **Contrastive representation learning** learns a shared embedding space where problem instances are matched to seed-generation methods based on utility supervision. (b) **Two-stage generation** first produces a seed in *No-Thinking* mode and then generates the final reasoning trace conditioned on the seed in *Thinking* mode. (c) **Token-level alignment self-enhanced speculative decoding** reuses the seed as a cost-free draft corpus, exploiting natural alignment between seed tokens and the reasoning trajectory to accelerate decoding.

Token saving. We measure inference cost by the total number of generated tokens, including both seed generation and final reasoning, denoted by $T(g, q)$. To obtain an interpretable efficiency score, we compare each seed method against the standard *Think* mode on the same problem. Let $T_{\text{Think}}(q)$ be the token cost of *Think*. We define the relative token-saving score as

$$S(g, q) = \max\left\{0,\ 1 - \frac{T(g, q)}{T_{\text{Think}}(q)}\right\}. \tag{2}$$

Thus, $S(g, q) = 0.4$ means that method g reduces token usage by 40% compared with *Think*.

Utility. We define the utility of seed method g on problem q as

$$U(g, q) = \lambda A(g, q) + (1 - \lambda) S(g, q), \quad (3)$$

where $\lambda \in [0, 1]$ controls the trade-off between solution correctness and token saving. Larger λ emphasizes correctness, while smaller λ gives more weight to efficiency.

The optimal seed-generation method for problem q is then defined as

$$g^*(q) = \arg\max_{g \in \mathcal{G}} U(g, q). \quad (4)$$

This utility provides a direct learning signal for seed-method selection. In the following section, we describe how to train a lightweight selector that predicts $g^*(q)$ for a new problem by matching problem representations with seed-method prototypes in a shared embedding space.

4.2 Contrastive Representation Learning

Problem and method representations. Let $\mathcal{D} = \{(q_j, \mathbf{f}_j)\}_{j=1}^N$ be the training set, where each problem q_j is associated with a feature vector $\mathbf{f}_j \in \mathbb{R}^D$. We learn a shared embedding space $\mathbb{R}^d$ that supports *matching* problems to seed-generation methods. A lightweight encoder $E_\theta : \mathbb{R}^D \to \mathbb{R}^d$ maps features to a problem embedding $\mathbf{p}_j = \mathrm{norm}(E_\theta(\mathbf{f}_j))$, where $\mathrm{norm}(\mathbf{x}) = \mathbf{x}/\|\mathbf{x}\|_2$ denotes ℓ_2 normalization. Each seed method $g_i \in \mathcal{G}$ is represented by a trainable prototype (method embedding) $\mathbf{e}_i \in \mathbb{R}^d$ with $\|\mathbf{e}_i\|_2 = 1$. We score compatibility as

$$s(\mathbf{p}, \mathbf{e}) = \mathbf{p}^\top \mathbf{e}. \quad (5)$$

This formulation yields a simple *prototype-based* selector with only m method vectors and a small encoder.

Utility-aware supervision. For each training problem q_j, we compute the utility $U(g_i, q_j)$ for all $g_i \in \mathcal{G}$ using Eq. (3), and define the optimal method label

$$i^*(j) = \arg\max_{i \in [m]} U(g_i, q_j), \quad (6)$$

where $[m] = \{1, \ldots, m\}$. This provides a direct supervision signal grounded in the desired accuracy–efficiency trade-off.

Hard-label InfoNCE objective. We first present a hard-label contrastive objective that pulls each problem toward its best method and pushes it away from all others:

$$\mathcal{L}_{\mathrm{NCE}} = -\frac{1}{N} \sum_{j=1}^{N} \log \frac{\exp\big(s(\mathbf{p}_j, \mathbf{e}_{i^*(j)})/\tau\big)}{\sum_{k=1}^{m} \exp(s(\mathbf{p}_j, \mathbf{e}_k)/\tau)}, \quad (7)$$

where $\tau > 0$ is a temperature parameter. Eq. (7) is equivalent to multiclass logistic regression over method prototypes, but written in a contrastive form that emphasizes geometric alignment.

Soft utility distillation. In practice, multiple methods can be near-optimal on the same problem. To reflect this, we further distill *relative utilities* into a soft target distribution. Define a utility-induced label distribution

$$\pi_j(i) = \frac{\exp(U(g_i, q_j)/\beta)}{\sum_{k=1}^{m} \exp(U(g_k, q_j)/\beta)}, \quad (8)$$

where $\beta > 0$ controls how sharply we emphasize the top methods. We then optimize a KL-matching objective against the model's predictive distribution $P_\theta(i \mid q_j) = \frac{\exp(s(\mathbf{p}_j, \mathbf{e}_i)/\tau)}{\sum_{k=1}^{m} \exp(s(\mathbf{p}_j, \mathbf{e}_k)/\tau)}$. The resulting loss is

$$\mathcal{L}_{\mathrm{soft}} = \frac{1}{N} \sum_{j=1}^{N} \mathrm{KL}(\pi_j(\cdot) \,\|\, P_\theta(\cdot \mid q_j)). \quad (9)$$

We use $\mathcal{L}_{\mathrm{soft}}$ as the default objective and recover Eq. (7) as the special case $\beta \to 0$. We include a standard noise- and gap-based stability argument in Appendix A to explain why utility-induced soft targets are preferable when multiple seed methods are near-optimal.

Overall training objective. Optionally, we regularize prototype geometry to avoid collapse and encourage separation:

$$\mathcal{L} = \mathcal{L}_{\mathrm{soft}} + \gamma \cdot \frac{1}{m(m-1)} \sum_{i \neq k} \left(\mathbf{e}_i^\top \mathbf{e}_k\right)^2, \quad (10)$$

where $\gamma \geq 0$ controls the strength of inter-prototype orthogonality regularization. We jointly optimize θ and $\{\mathbf{e}_i\}_{i=1}^m$ via SGD.

4.3 Inference-Time Routing and Two-Stage Generation

Given a problem q, we extract features $\mathbf{f}_q$ and compute the embedding $\mathbf{p}_q = \mathrm{norm}(E_\theta(\mathbf{f}_q))$. We then

route q to a seed-generation method by nearest-prototype matching:

$$g_{\text{sel}}(q) = \arg\max_{g_i \in \mathcal{G}} s(\mathbf{p}_q, \mathbf{e}_i). \tag{11}$$

Since $|\mathcal{G}|$ is small, routing incurs negligible overhead (one encoder forward pass and $|\mathcal{G}|$ dot products). We then run a two-stage generation pipeline with the same base model M: (i) *seed drafting* in the *No-Thinking* mode under the selected template, and (ii) *seed-guided reasoning* in the *Thinking* mode conditioned on the question and the seed-thought:

$$s_{seed} = M(g_{\text{sel}}, q)\,, \tag{12}$$

$$r_{\text{SeedThink}} = M(q, s_{seed})\,. \tag{13}$$

4.4 Self-Enhanced Speculative Decoding

SeedThink is designed to unlock *an additional acceleration route* beyond shortening the reasoning trace itself: we *reuse the seed-thought* as an internal, cost-free, accurate draft for speculative decoding. Concretely, the two-stage pipeline in Eq. (12)–(13) first produces a seed s and then generates the final reasoning trace r conditioned on (q, s) using the same base model M. **This shared-model, seed-conditioned generation induces *token-level alignment* between s and r: many seed tokens (and short n-grams) tend to reappear in the subsequent reasoning, making s a high-quality speculative draft.** Unlike conventional speculative decoding that relies on a separate draft model, our draft is *model-free*—it is generated once by M in *No-Thinking* mode and then statically reused to propose tokens for the second-stage *Thinking* generation, which does not introduce additional decoding overhead. When the verifier (the same model M) accepts proposed draft tokens from a seed, we skip its expensive generation step, yielding extra speedups on top of token reduction.

The following proposition formalizes this mechanism by lower-bounding the probability that seed n-grams are reproduced by the reasoning trace (proof in Appendix B).

Proposition (Seed–Reasoning Token Alignment)**.** *For any seed n-gram $g = s_{i:i+n-1}$, the probability that the reasoning trace reproduces g at the matched position satisfies*

$$\Pr[r_{i:i+n-1} = g] \;\geq\; \prod_{t=i}^{i+n-1} \big(1 - \varepsilon_t(s_{<t})\big),$$

where $\varepsilon_t(u) = \|p_s(\cdot \mid u) - p_r(\cdot \mid u)\|_{\text{TV}}$.

When the two next-token distributions are close so that ε_t is small, this bound implies a high probability of token reuse. A corresponding "anywhere" version over a candidate position set follows by a union bound (Appendix B). We empirically validate this effect in Section 5.5 using the mean number of accepted tokens, which directly measures how many tokens are reused during speculative decoding.

5 EVALUATION

5.1 Experimental Setup

Datasets. We use the MATH (Hendrycks et al., 2021) dataset as a training set, utilizing its training split of 7,500 math problems with solutions. We evaluate on two math benchmarks: **GSM8K** (Cobbe et al., 2021), **MATH500**, and one finance dataset **Fineval** (Guo et al., 2025b). MATH500 is a test split of MATH, which contains 500 samples, as defined in (Lightman et al., 2023). We report **Pass@1** accuracy, speed up based on wall-clock time, and response length (tokens). All answers are verified using HuggingFace Math-Verify.[1]

Models. We use five public LRMs: GPT-OSS-20B (Agarwal et al., 2025), DeepSeek-R1-Distill-Llama-8B (Guo et al., 2025a), and Qwen3-8B/14B/32B (Yang et al., 2025).

Baselines. We compare our approach against four representative baseline methods:

- *Think:* The standard *Thinking* mode, where the model generates a full, multi-step reasoning trace before producing a final answer.
- *NoThink:* The standard *No-Thinking* mode, where the model generates a direct, concise answer without an explicit reasoning trace.
- *JointThink (Wu et al., 2025a):* A calibration-based method that first generates answers in parallel using both *Thinking* and *No-Thinking* modes. If the answers from two modes are inconsistent, it triggers a second *Thinking* round conditioned on both candidates' answers to improve robustness.
- *CoThink (Fan et al., 2025):* A two-stage pipeline where an instruct model first creates a very short reasoning outline, which is then refined by a reasoning model.

Additional implementation details and experimental settings are provided in Appendix E.

[1] https://github.com/huggingface/Math-Verify

Table 1: **Main results on MATH500.** Accuracy, average response length, and runtime speedup relative to *Think*. Reported token counts include seed generation. Utility is computed by Eq. 3 with $\lambda = 0.5$.

	GPT-OSS-20B				*Qwen3-14B*				*Qwen3-32B*			
Method	Acc.↑	Speedup↑	Len.↓	Utility↑	Acc.↑	Speedup↑	Len.↓	Utility↑	Acc.↑	Speedup↑	Len.↓	Utility↑
Think	97.00%	1.0×	4,380	-	96.80%	1.0×	4,741	-	97.20%	1.0×	4,657	-
NoThink	89.20%	6.3×	646	-	86.40%	5.1×	895	-	86.40%	6.0×	774	-
JointThink	96.80%	0.7×	6,418	0.484	97.20%	0.7×	6,273	0.486	97.60%	0.6×	6,524	0.488
CoThink	94.80%	1.2×	3,867	0.533	93.80%	1.5×	2,959	0.657	94.60%	1.6×	3,137	0.636
SeedThink (ours)	95.20%	2.6×	3,325	**0.596**	94.60%	2.9×	2,761	**0.682**	95.20%	3.1×	2,559	**0.701**

5.2 Main Results

Efficiency–accuracy trade-off on MATH500. Table 1 reports results on MATH500, where the selector is trained only on the MATH training set. Across GPT-OSS-20B and Qwen3 models, SeedThink consistently improves efficiency over *Think*, achieving 2.6–3.1× speedups with substantially shorter generations. Compared with CoThink, SeedThink achieves higher accuracy, lower runtime, shorter length, and better overall utility under Eq. (3). This confirms the benefit of adaptive seeding: instead of using a fixed outline, SeedThink selects the seed granularity per problem to balance guidance and flexibility. JointThink can slightly improve accuracy in some cases, but its longer traces make it less favorable under compute constraints.

Complementary sources of speedup. SeedThink accelerates inference through both trajectory shortening and faster token generation. Seed-conditioned reasoning reduces unnecessary exploration, while self-enhanced speculative decoding reuses the seed as a compatible draft for the subsequent reasoning stage. In Table 1, SeedThink obtains 2.6–3.1× speedups while reducing generation length by about 24%–45%, indicating additional gains beyond length reduction alone.

5.3 The Impact of the Trade-off Parameter λ

The parameter λ in Eq. (3) controls the preference between solution correctness and inference cost when selecting seed-generation methods. We use $\lambda = 0.5$ as a balanced default, while allowing it to be adjusted for accuracy-critical or cost-sensitive deployments. A sensitivity analysis shows that varying λ leads to smooth and predictable shifts in the accuracy–efficiency trade-off, with the main gains of adaptive seeding remaining stable. Detailed results are reported in Appendix H.

5.4 Generalization and Robustness

We evaluate transferability across unseen math benchmarks, professional non-math domains, and batched inference settings.

Table 2: **Performance on In-domain dataset GSM8K.** The selector is trained on the MATH dataset and directly applied to GSM8K without additional training. Utility is computed by Eq. 3 with $\lambda = 0.5$.

Method	Accuracy↑	Speedup↑	Length↓	Utility↑
DeepSeek-R1-Distill-LLaMA-8B				
Think	76.50%	1.0×	4,029	-
NoThink	72.80%	9.5×	320	-
JointThink	79.30%	0.8×	4,968	0.397
CoThink	77.64%	1.4×	3,133	0.499
SeedThink (ours)	78.22%	2.6×	2,847	**0.538**
Qwen3-32B				
Think	95.30%	1.0×	1,731	-
NoThink	91.52%	5.5×	286	-
JointThink	95.83%	0.8×	2,243	0.479
CoThink	94.62%	2.4×	820	0.736
SeedThink (ours)	94.84%	4.1×	560	**0.812**

Table 3: **Performance on the out-of-domain professional finance dataset FinEval.** Utility is computed by Eq. 3 with $\lambda = 0.5$.

Method	Accuracy↑	Speedup↑	Length↓	Utility↑
Think	68.64%	1.0×	2,606	-
NoThink	58.30%	5.3×	498	-
CoThink	64.29%	1.4×	1,926	0.452
SeedThink (ours)	67.94%	2.3×	1,814	**0.492**

5.4.1 In-domain Transferability

Table 2 evaluates transfer from MATH500 to GSM8K. Although the selector is trained only on MATH, SeedThink preserves its efficiency advantage across model families. On DeepSeek-R1-Distill-LLaMA-8B, SeedThink improves accuracy over *Think* (78.22% vs. 76.50%) while achieving a 2.6× speedup. On Qwen3-32B, it achieves a 4.1× speedup and reduces length from 1,731 to 560 tokens with only a small accuracy change. We further report AIME 2024 results in Appendix F, where SeedThink remains effective on harder math problems with larger *Think*/*NoThink* gaps. These results suggest that the learned problem–seed matching captures transferable cues about how much guidance a problem needs, rather than overfitting to MATH500.

Table 4: **Batched inference results without speculative decoding.** Results are reported on Qwen3-32B with disabled SD. Utility is computed by Eq. 3 with $\lambda = 0.5$.

Method	Accuracy↑	Speedup↑	Length↓	Utility↑
Think	97.60%	1.0×	4,299	-
NoThink	88.00%	2.9×	742	-
CoThink	94.40%	1.4×	3,289	0.589
SeedThink (ours)	94.80%	1.8×	2,512	**0.682**

5.4.2 Out-of-domain Transferability

Table 3 reports results on FinEval, a professional finance benchmark. SeedThink achieves a 2.3× speedup and shortens generation from 2,606 to 1,814 tokens, while maintaining accuracy close to *Think* (67.94% vs. 68.64%). This suggests that seed initialization remains useful even when the task requires domain-specific knowledge rather than purely mathematical manipulation. It also outperforms CoThink in both accuracy and efficiency, indicating that adaptive seed selection provides more effective guidance than a fixed outline-style scaffold. Additional results on GPQA-Diamond are provided in Appendix G, showing that seed-based acceleration also transfers to scientific reasoning.

5.4.3 Batched Non-SD Inference

We evaluate SeedThink under batched inference with speculative decoding (SD) disabled, in which gains must come from trajectory optimization alone. As shown in Table 4, SeedThink still reduces runtime and length while preserving competitive accuracy. This setting isolates the contribution of seed-conditioned reasoning from the additional acceleration introduced by speculative decoding. The results confirm that SeedThink's benefits are not limited to SD, but also arise from more focused and stable reasoning trajectories.

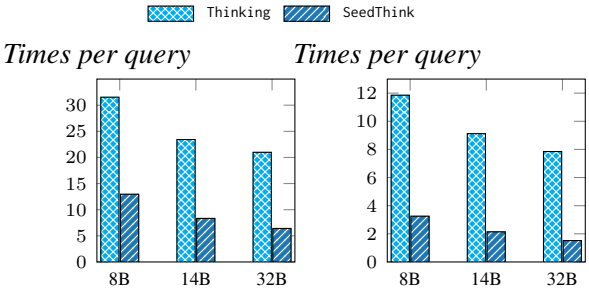
5.5 Ablation Study on Speculative Decoding

We ablate whether SeedThink's speculative gains come from merely adding draft tokens or from the path dependence induced by seed-conditioned reasoning. We compare standard speculative decoding on *Think* traces with a warm-up variant that uses the seed only as an external SAM draft, without conditioning final reasoning on it. This comparison separates the value of seed reuse as an external draft corpus from the value of actually initializing the reasoning trajectory with the seed.

Table 5 shows that warm-up improves decoding but remains weaker than SeedThink. Conditioning reasoning on the seed yields higher mean accepted tokens (MAT), indicating stronger token-level alignment between the draft and the final trajectory. This supports our claim that SeedThink's speculative benefit comes from seed-induced trajectory alignment rather than simply increasing the amount of available draft text. Further analysis of seed length is deferred to Appendix I.

Table 5: Ablation study on MATH500 using Qwen3-4B to compare SD performance. MAT: Mean Accepted Tokens; Throughput: tokens per second. Speedups are normalized to the Think baseline.

Method	**MAT↑**	**Throughput↑**	**Speed up↑**
Think	1.00	28.4	–
+ SAM	1.75	38.8	1.37×
+ SAM + warmup	1.88	41.2	1.45×
+ SAM + Eagle3	3.26	54.5	1.92×
+ all components	3.41	59.0	2.08×
SeedThink	1.00	28.6	1.00×
+ SAM	1.94	44.2	1.56×
+ SAM + Eagle3	3.65	63.6	2.24×



(a) *Reflection* (b) *Branch exploration*

Figure 4: SeedThink reduces inefficient reasoning patterns across model scales. We report the average number of (a) reflection events and (b) branch-exploration events per query for Thinking versus SeedThink on Qwen3-8B, 14B, and 32B. SeedThink consistently suppresses both behaviors, with larger reductions on larger models.

5.6 Trajectory Analysis

Figure 4 shows that SeedThink reduces two common inefficiency patterns in LRM traces: reflection and branching. Across Qwen3 models, it reduces reflection by 58.9%–69.5% and branching by 72.6%–80.6%. This indicates that seed initialization stabilizes the reasoning trajectory and avoids unnecessary self-verification and low-yield detours. Detailed definitions are provided in Appendix J.

6 Conclusion

We presented SeedThink, a unified framework for efficient reasoning in LRMs that adaptively selects and exploits seed thoughts. SeedThink produces focused reasoning trajectories and enables an additional acceleration path through token-level alignment for speculative decoding.

Limitations

This work studies efficient test-time reasoning with seed-thought initialization. Although our experiments cover mathematical, financial, and scientific reasoning benchmarks, broader evaluation on tasks such as code generation, multi-hop question answering, and interactive tool-use remains an important direction. Our current implementation focuses on a compact set of seed granularities, which enables controlled analysis of the guidance–flexibility trade-off. Future work may explore automatic seed-strategy discovery and finer-grained adaptation.

Ethics Statement

This work focuses on improving the inference efficiency of reasoning-capable language models. We do not introduce new datasets, collect user data, involve human subjects, or train new foundation models. All experiments are conducted using publicly available benchmark datasets and widely used open-source language models. The proposed method operates at inference time and aims to reduce computational cost while preserving reasoning quality.

The potential risks of this work are generally aligned with those of existing large language models, such as possible misuse, bias in model outputs, or overreliance on generated reasoning. Since Seed-Think improves efficiency, it may lower the cost of deploying reasoning models; therefore, responsible deployment practices, including task-appropriate evaluation and human oversight in high-stakes applications, remain important.

Use of AI assistants. We used AI assistants for writing support, including grammar correction, language polishing, LaTeX formatting, and improving clarity of presentation. We also used AI assistants for limited coding assistance and debugging during experimental implementation. All research ideas, method design, experimental results, analyses, and final manuscript content were reviewed, verified, and approved by the authors. AI assistants were not used as authors and did not make independent scientific contributions.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Qiguang Chen, Dengyun Peng, Jinhao Liu, HuiKang Su, Jiannan Guan, Libo Qin, and Wanxiang Che. 2025. Aware first, think less: Dynamic boundary self-awareness drives extreme reasoning efficiency in large language models. *arXiv preprint arXiv:2508.11582*.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, and 1 others. 2024. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, and 1 others. 2025. The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks. *arXiv preprint arXiv:2502.08235*.

Siqi Fan, Peng Han, Shuo Shang, Yequan Wang, and Aixin Sun. 2025. Cothink: Token-efficient reasoning via instruct models guiding reasoning models. *arXiv preprint arXiv:2505.22017*.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. Thinkless: Llm learns when to think. *arXiv preprint arXiv:2505.13379*.

Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Aurick Qiao, and Hao Zhang. 2024. Efficiently serving llm reasoning programs with certaindex. *arXiv preprint arXiv:2412.20993*.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 2024. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025a. Deepseek-r1: Incentivizing reasoning capability in

llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Xin Guo, Haotian Xia, Zhaowei Liu, Hanyang Cao, Zhi Yang, Zhiqiang Liu, Sizhe Wang, Jinyi Niu, Chuqi Wang, Yanhui Wang, and 1 others. 2025b. Fineval: A chinese financial domain knowledge evaluation benchmark for large language models. 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 6258–6292.

Rui Ha, Chaozhuo Li, Rui Pu, and Sen Su. 2025. From" aha moments" to controllable thinking: Toward meta-cognitive reasoning in large reasoning models via decoupled reasoning and control. *arXiv preprint arXiv:2508.04460*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, and 1 others. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. 2023. Rest: Retrieval-based speculative decoding. *arXiv preprint arXiv:2311.08252*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. 2025. Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning. *arXiv preprint arXiv:2504.01296*.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. 2024. Sam decoding: Speculative decoding via suffix automaton. *arXiv preprint arXiv:2411.10666*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2023. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. *arXiv preprint arXiv:2302.09664*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proc. Int. Conf. Machine Learning*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857.

Yang Li, Youssef Emad, Karthik Padthe, Jack Lanchantin, Weizhe Yuan, Thao Nguyen, Jason Weston, Shang-Wen Li, Dong Wang, Ilia Kulikov, and 1 others. 2025a. Naturalthoughts: Selecting and distilling reasoning traces for general reasoning tasks. *arXiv preprint arXiv:2507.01921*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025b. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*.

Guosheng Liang, Longguang Zhong, Ziyi Yang, and Xiaojun Quan. 2025. Thinkswitcher: When to think hard, when to think fast. *arXiv preprint arXiv:2505.14183*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Zhengkai Lin, Zhihang Fu, Ze Chen, Chao Chen, Liang Xie, Wenxiao Wang, Deng Cai, Zheng Wang, and Jieping Ye. 2025. Controlling thinking speed in reasoning models. *arXiv preprint arXiv:2507.03704*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Haotian Luo, Haiying He, Yibo Wang, Jinluan Yang, Rui Liu, Naiqiang Tan, Xiaochun Cao, Dacheng Tao, and Li Shen. 2025. Ada-r1: Hybrid-cot via bi-level adaptive reasoning optimization. *arXiv preprint arXiv:2504.21659*.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. 2024. Turning trash into treasure: Accelerating inference of large language models with token recycling. *arXiv preprint arXiv:2408.08696*.

Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. 2025. Reasoning models can be effective without thinking. *arXiv preprint arXiv:2504.09858*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, and 1 others. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference

and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. 2024. Skeleton-of-thought: Prompting llms for efficient parallel generation. In *International Conference on Learning Representations*, volume 2024, pages 917–967.

Gabriele Oliaro, Zhihao Jia, Daniel F Campos, and Aurick Qiao. 2025. Suffixdecoding: Extreme speculative decoding for emerging ai applications. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. 2025. Specreason: Fast and accurate inference-time compute via speculative reasoning. *arXiv preprint arXiv:2504.07891*.

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

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. *arXiv preprint arXiv:2410.20290*.

Shengyin Sun, Yiming Li, Xing Li, Yingzhao Lian, Weizhe Lin, Hui-Ling Zhen, Zhiyuan Yang, Chen Chen, Xianzhi Yu, Mingxuan Yuan, and 1 others. 2025. Scaling up, speeding up: A benchmark of speculative decoding for efficient llm test-time scaling. *arXiv preprint arXiv:2509.04474*.

Chenlong Wang, Yuanning Feng, Dongping Chen, Zhaoyang Chu, Ranjay Krishna, and Tianyi Zhou. 2025a. Wait, we don't need to" wait"! removing thinking tokens improves reasoning efficiency. *arXiv preprint arXiv:2506.08343*.

Jikai Wang, Juntao Li, Lijun Wu, and Min Zhang. 2025b. Efficient reasoning for llms through speculative chain-of-thought. *arXiv preprint arXiv:2504.19095*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 2609–2634.

Yiming Wang, Pei Zhang, Siyuan Huang, Baosong Yang, Zhuosheng Zhang, Fei Huang, and Rui Wang. 2025c. Sampling-efficient test-time scaling: Self-estimating the best-of-n sampling in early decoding. *arXiv preprint arXiv:2503.01422*.

Zhihai Wang, Jie Wang, Jilai Pan, Xilin Xia, Huiling Zhen, Mingxuan Yuan, Jianye Hao, and Feng Wu. 2025d. Accelerating large language model reasoning via speculative search. *arXiv preprint arXiv:2505.02865*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Haotian Wu, Bo Xu, Yao Shu, Menglin Yang, and Chengwei Qin. 2025a. Thinking with nothinking calibration: A new in-context learning paradigm in reasoning large language models. *arXiv preprint arXiv:2508.03363*.

Tong Wu, Chong Xiang, Jiachen T Wang, G Edward Suh, and Prateek Mittal. 2025b. Effectively controlling reasoning models through thinking intervention. *arXiv preprint arXiv:2503.24370*.

Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*.

Zeyu Xing, Xing Li, Huiling Zhen, Xianzhi Yu, Mingxuan Yuan, and Sinno Jialin Pan. Large reasoning models know how to think efficiently. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. 2025a. Adaptthink: Reasoning models can learn when to think. *arXiv preprint arXiv:2505.13417*.

Shengjia Zhang, Junjie Wu, Jiawei Chen, Changwang Zhang, Xingyu Lou, Wangchunshu Zhou, Sheng Zhou, Can Wang, and Jun Wang. 2025b. Othink-r1: Intrinsic fast/slow thinking mode switching for over-reasoning mitigation. *arXiv preprint arXiv:2506.02397*.

Wencheng Zhang, Shiqin Qiao, Lingjie Luo, Yinfeng Li, Chuanyang Zheng, Qian Xu, Meng Li, Yong Gui, Yijun He, Jianing Qiu, and 1 others. 2025c.

Synapseroute: An auto-route switching framework on dual-state large language model. *arXiv preprint arXiv:2507.02822*.

Yao Zhao, Zhitian Xie, Chen Liang, Chenyi Zhuang, and Jinjie Gu. 2024. Lookahead: An inference acceleration framework for large language model with lossless generation accuracy. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6344–6355.

Appendix Roadmap

For ease of navigation, we provide a roadmap of the supplementary materials.

Section	Content
App. A	Formal justification for soft utility distillation.
App. B	Proof of the token-alignment guarantee for seed reuse.
App. C	Supplementary materials for the motivating observation in Section 3.2.
App. D	Implementation details for contrastive representation learning.
App. E	Supplementary details for experimental setup.
App. F	Additional in-domain transfer results on AIME 2024.
App. G	Additional out-of-domain evaluation on GPQA-Diamond.
App. H	Sensitivity analysis of the trade-off parameter λ.
App. I	Effect of seed length on speculative acceptance and efficiency.
App. J	Details of reasoning trajectory optimization analysis.
App. K	Prompt templates used for seed generation.

A A More Formal Justification for Soft Utility Distillation

A.1 Setup: Utility, Noise, and Targets

Fix a problem instance q. Let $U_i := U(g_i, q)$ denote the true utility of method $g_i \in \mathcal{G}$ for $i \in [m]$. In practice, utilities are estimated from stochastic decoding and finite evaluation, so we observe

$$\widehat{U}_i = U_i + \xi_i, \tag{14}$$

where ξ_i denotes the estimation noise.

We compare two supervision signals for training the selector. The hard-label target selects the empirically best method:

$$Y = \arg\max_{i \in [m]} \widehat{U}_i, \qquad y_i = \mathbb{I}\{i = Y\}. \tag{15}$$

The soft utility target instead converts the estimated utilities into a distribution:

$$\pi_\beta(i) = \frac{\exp(\widehat{U}_i/\beta)}{\sum_{k=1}^{m} \exp(\widehat{U}_k/\beta)}, \qquad \beta > 0. \tag{16}$$

The selector predicts a distribution induced by prototype matching:

$$P_\theta(i \mid q) = \frac{\exp(s(\mathbf{p}, \mathbf{e}_i)/\tau)}{\sum_{k=1}^{m} \exp(s(\mathbf{p}, \mathbf{e}_k)/\tau)}. \tag{17}$$

A.2 Hard Labels Are Discontinuous; Soft Targets Are Lipschitz

We first show that the mapping from estimated utilities to a hard label is discontinuous, while the soft target changes smoothly with utility perturbations.

Lemma A.1 (Softmax Lipschitz in ℓ_1)**.** *Let $\sigma_\beta(\mathbf{u}) = \mathrm{softmax}(\mathbf{u}/\beta)$ for $\mathbf{u} \in \mathbb{R}^m$ and $\beta > 0$. Then for any $\mathbf{u}, \mathbf{v} \in \mathbb{R}^m$,*

$$\begin{aligned} \|\sigma_\beta(\mathbf{u}) - \sigma_\beta(\mathbf{v})\|_1 &\le \frac{1}{2\beta}\|\mathbf{u} - \mathbf{v}\|_1 \\ &\le \frac{m}{2\beta}\|\mathbf{u} - \mathbf{v}\|_\infty. \end{aligned} \tag{18}$$

Proof. Let $\mathbf{p} = \sigma_\beta(\mathbf{u})$. The Jacobian is

$$\nabla\sigma_\beta(\mathbf{u}) = \frac{1}{\beta}\big(\mathrm{Diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top\big). \tag{19}$$

For any coordinate j, we have

$$\begin{aligned} \sum_{i=1}^{m} \left|\frac{\partial p_i}{\partial u_j}\right| &= \frac{1}{\beta}\sum_{i=1}^{m} \big|p_i\big(\mathbb{I}\{i = j\} - p_j\big)\big| \\ &= \frac{2p_j(1 - p_j)}{\beta} \le \frac{1}{2\beta}. \end{aligned} \tag{20}$$

Thus $\|\nabla\sigma_\beta(\mathbf{u})\|_{1\to1} \le 1/(2\beta)$. Applying the mean value theorem along the segment between $\mathbf{u}$ and $\mathbf{v}$ gives the first inequality in Eq. (18). The second follows from $\|\mathbf{x}\|_1 \le m\|\mathbf{x}\|_\infty$. □

Proposition A.2 (Hard-label instability under small utility gaps)**.** *Let $i^\star = \arg\max_i U_i$ be the true best method and define*

$$\delta = U_{i^\star} - \max_{i \ne i^\star} U_i \ge 0. \tag{21}$$

Assume $|\xi_i| \le \varepsilon$ almost surely. If $\delta \le 2\varepsilon$, then admissible noise realizations can change the hard label. In contrast, the soft target $\pi_\beta = \sigma_\beta(\widehat{\mathbf{U}})$ changes continuously with $\widehat{\mathbf{U}}$ and satisfies Lemma A.1.

Proof. Let j be an index satisfying $U_j = \max_{i \ne i^\star} U_i$, so $U_{i^\star} - U_j = \delta$. Choose $\xi_{i^\star} = -\varepsilon$ and $\xi_j = +\varepsilon$. Then

$$\widehat{U}_{i^\star} - \widehat{U}_j = \delta - 2\varepsilon \le 0, \tag{22}$$

so the maximizer can flip from $i^\star$ to j. Choosing the opposite signs recovers $i^\star$ as the maximizer. The continuity and Lipschitz property of the soft target follow from Lemma A.1. □

A.3 A Regret-Oriented View: Avoid Clearly Bad Methods

In seed selection, multiple seed methods can be near-optimal for the same problem. Hard one-hot supervision forces the selector to distinguish among these near-ties, although this distinction is often unnecessary from a regret perspective.

Proposition A.3 (Regret and near-optimal sets)**.** *Fix q and let $U^\star = \max_i U_i$. Suppose there exists a near-optimal set $\mathcal{S} \subseteq [m]$ such that*

$$U^\star - U_i \le \Delta, \qquad \forall i \in \mathcal{S}. \tag{23}$$

Let

$$R_{\max} = \max_{k \notin \mathcal{S}}\big(U^\star - U_k\big), \qquad 0 \le \Delta \le R_{\max}. \tag{24}$$

Then for any selection distribution $P(\cdot \mid q)$,

$$\mathbb{E}_{i\sim P}[U^\star - U_i] \le \Delta + (R_{\max} - \Delta)P(i \notin \mathcal{S} \mid q). \tag{25}$$

Proof. We decompose the expected regret over $\mathcal{S}$ and its complement:

$$\begin{aligned} &\mathbb{E}_{i\sim P}[U^\star - U_i] \\ &= \sum_{i\in\mathcal{S}} P(i)(U^\star - U_i) + \sum_{k\notin\mathcal{S}} P(k)(U^\star - U_k) \\ &\le \Delta \sum_{i\in\mathcal{S}} P(i) + R_{\max} \sum_{k\notin\mathcal{S}} P(k) \\ &= \Delta + (R_{\max} - \Delta)P(i \notin \mathcal{S} \mid q). \end{aligned} \tag{26}$$

This proves Eq. (25). □

Implication. When several methods are near-optimal, the main source of regret is not how probability mass is distributed inside $\mathcal{S}$, but how much mass is assigned outside $\mathcal{S}$. Soft utility distillation naturally matches this structure by assigning similar target probabilities to methods with similar utilities, while suppressing clearly suboptimal methods.

A.4 Noise-Induced Label Error

We further quantify the probability that noisy utility estimates produce an incorrect hard label.

Proposition A.4 (Mislabel probability under sub--Gaussian noise)**.** *Assume $\{\xi_i\}_{i=1}^m$ are independent, mean-zero, and σ^2-sub-Gaussian. Let $i^\star = \arg\max_i U_i$ and let δ be defined in Eq. (21). Then the hard label $Y = \arg\max_i \widehat{U}_i$ satisfies*

$$\mathbb{P}(Y \ne i^\star) \le (m - 1)\exp\left(-\frac{\delta^2}{8\sigma^2}\right). \tag{27}$$

Proof. For any $j \neq i^\star$, define $E_j = \{\widehat{U}_j \geq \widehat{U}_{i^\star}\}$. By the union bound,

$$\mathbb{P}(Y \neq i^\star) \leq \sum_{j \neq i^\star} \mathbb{P}(E_j). \tag{28}$$

The event E_j implies

$$\xi_j - \xi_{i^\star} \geq U_{i^\star} - U_j \geq \delta. \tag{29}$$

Since $\xi_j - \xi_{i^\star}$ is mean-zero and $2\sigma^2$-sub-Gaussian,

$$\mathbb{P}(E_j) \leq \exp\left(-\frac{\delta^2}{8\sigma^2}\right). \tag{30}$$

Summing over $j \neq i^\star$ yields Eq. (27). □

Implication. When the gap δ is small, the hard label is unstable and may inject label noise into selector training. In contrast, the soft target $\pi_\beta = \sigma_\beta(\widehat{\mathbf{U}})$ changes smoothly with noisy utility estimates, as shown in Lemma A.1.

Takeaway. Propositions A.2–A.4 formalize why soft utility distillation is suitable for seed-method selection: it is continuous under noisy utility estimates and respects the regret structure in which assigning probability mass among near-optimal methods is acceptable, while clearly suboptimal methods should be suppressed.

B Proof of Proposition 4.4

B.1 Preliminaries

Let M be the same autoregressive model used in both stages. Let $s = (s_1, \ldots, s_k)$ be the seed generated in the first stage, and let $r = (r_1, r_2, \ldots)$ be the reasoning trace generated in the second stage conditioned on (q, s). Let $\mathcal{V}$ denote the vocabulary.

For any prefix string u, define two next-token distributions: $p_s(\cdot \mid u)$ under the seed-generation context, and $p_r(\cdot \mid u)$ under the seed-guided reasoning context. Their total variation distance is

$$\begin{aligned}\varepsilon(u) &:= \|p_s(\cdot \mid u) - p_r(\cdot \mid u)\|_{\mathrm{TV}} \\ &= \frac{1}{2}\sum_{x \in \mathcal{V}} |p_s(x \mid u) - p_r(x \mid u)|.\end{aligned} \tag{31}$$

For prefixes encountered during reasoning, define the expected stepwise divergence as

$$\bar{\varepsilon}_t = \mathbb{E}_{r_{<t} \sim P_r}\left[\varepsilon(r_{<t})\right]. \tag{32}$$

When the seed-generation and seed-conditioned reasoning contexts induce close next-token distributions, we assume there exists a constant ε such that

$$\bar{\varepsilon}_t \leq \varepsilon, \qquad \forall t. \tag{33}$$

This condition can be viewed as a local alignment assumption between the two decoding stages.

B.2 Matched-Position n-Gram Alignment

Proposition (Matched-position alignment)**.** *Fix any seed n-gram $g = s_{i:i+n-1}$ with $1 \leq i \leq i+n-1 \leq k$. If the reasoning trace follows the same prefix up to the matched position, then*

$$\begin{aligned}\Pr[r_{i:i+n-1} = g] &\geq \prod_{t=i}^{i+n-1} \big(1 - \varepsilon(s_{<t})\big) \\ &\geq (1-\varepsilon)^n \geq 1 - n\varepsilon.\end{aligned} \tag{34}$$

Proof. For each $t \in \{i, \ldots, i+n-1\}$, define

$$E_{t-1} := \{r_{i:t-1} = s_{i:t-1}\}. \tag{35}$$

On event E_{t-1}, the two processes share the same prefix up to step $t-1$, so $r_{<t} = s_{<t}$.

By the coupling characterization of total variation distance, for any prefix u there exists a coupling (X, Y) such that

$$X \sim p_s(\cdot \mid u), \qquad Y \sim p_r(\cdot \mid u), \tag{36}$$

and

$$\Pr[X \neq Y] = \|p_s(\cdot \mid u) - p_r(\cdot \mid u)\|_{\mathrm{TV}}. \tag{37}$$

Applying this coupling with $u = s_{<t}$ gives

$$\Pr[r_t = s_t \mid E_{t-1}] \geq 1 - \varepsilon(s_{<t}). \tag{38}$$

Using the chain rule,

$$\begin{aligned}&\Pr[r_{i:i+n-1} = s_{i:i+n-1}] \\ &= \prod_{t=i}^{i+n-1} \Pr[r_t = s_t \mid r_{i:t-1} = s_{i:t-1}] \\ &\geq \prod_{t=i}^{i+n-1} \big(1 - \varepsilon(s_{<t})\big).\end{aligned} \tag{39}$$

This proves the first inequality in Eq. (34). The remaining inequalities follow from $(1-\varepsilon)^n \geq 1 - n\varepsilon$ for $\varepsilon \in [0, 1)$. □

B.3 Corollary: Appearing at Any Candidate Position

The matched-position result can be extended to appearance within a candidate set of positions.

Corollary B.1 (Anywhere alignment over a candidate set)**.** *Fix a seed n-gram $g = s_{i:i+n-1}$ and a candidate set of starting positions J. Assume that*

for every $j \in J$ and every $t \in \{j, \ldots, j+n-1\}$, the corresponding stepwise divergence satisfies

$$\varepsilon(\widetilde{u}_t) \le \varepsilon, \tag{40}$$

where $\widetilde{u}_t$ is the shared prefix under the event that g is generated at position j. Then

$$\begin{aligned} &\Pr[\exists j \in J : r_{j:j+n-1} = g] \\ &\ge 1 - |J|\,(1-(1-\varepsilon)^n) \\ &\ge 1 - |J| n\varepsilon. \end{aligned} \tag{41}$$

Proof. For each $j \in J$, let

$$A_j = \{r_{j:j+n-1} = g\}. \tag{42}$$

By the union bound,

$$\begin{aligned} \Pr[\exists j \in J : A_j] &= 1 - \Pr\left[\bigcap_{j \in J} A_j^c\right] \\ &\ge 1 - \sum_{j \in J} \Pr[A_j^c]. \end{aligned} \tag{43}$$

Under Eq. (40), the same coupling and chain-rule argument as in Proposition B.2 gives

$$\Pr[A_j] \ge (1-\varepsilon)^n, \qquad \forall j \in J. \tag{44}$$

Therefore,

$$\Pr[A_j^c] \le 1 - (1-\varepsilon)^n. \tag{45}$$

Substituting this into Eq. (43) yields the first inequality in Eq. (41). The second follows from

$$1 - (1-\varepsilon)^n \le n\varepsilon. \tag{46}$$

□

Interpretation. The bound is unconditional and holds for arbitrary stepwise divergences. Its informativeness depends on the magnitude of ε along the seed prefix. When the seed-generation and reasoning-conditioned next-token distributions remain close over consecutive steps, the product form yields a non-trivial lower bound. This implies that seed tokens and short n-grams are likely to be reused in the subsequent reasoning trace. When the distributions diverge, the bound degrades gracefully. Thus, the result characterizes how token-level alignment accumulates across steps, while empirical measurements in Section 5.5 indicate when this alignment is substantial in practice.

C Supplementary Materials for Section 3.2

As described in Section 3.2, we conduct a pilot study using the MATH500 datasets to analyze the impact of seed granularity on the performance of two widely used LRMs Qwen3-8B and DeepSeek-R1-Distill-LLaMA-8B. For seed thought generation, we use the No-Thinking mode with four different prompt to control the seed granularity. For the second round of generation, we use the thinking mode while providing the seed as an additional reference through the thinking with seed prompt. Both the seed generation prompt and the thinking with seed prompt can be found in K.

We define four levels of seed granularity in this pilot study:

- *Outline seed:* High-level strategic breakdown focusing only on outline without numbers, formulas, or calculations
- *Key-steps seed:* Main reasoning steps with important formulas/concepts mentioned.
- *General-approach seed:* Step-by-step plan with brief explanations, including formulas and concepts, but avoiding extensive calculations
- *Detailed-Steps seed:* Near-complete solution with formulas, initial setups, and simple calculations, leaving only complex reasoning for completion

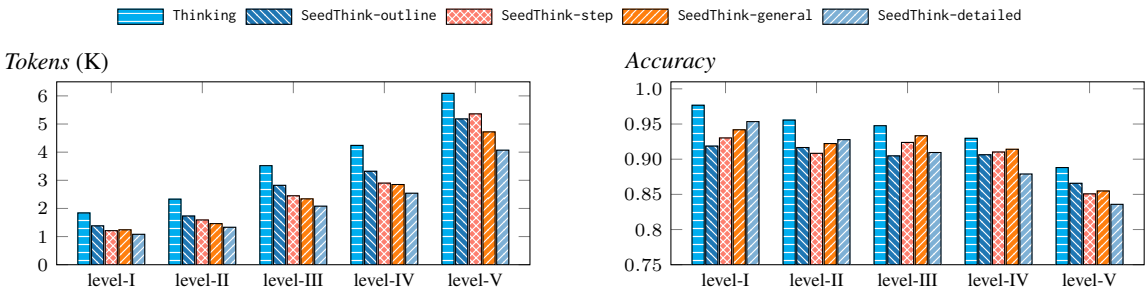
We use an 8K context window for seed-thought generation and 16K context window for thinking with seed reasoning process. Following the recommendations to balance diversity and coherence in the official model cards, we set the temperature to 0.6 for Thinking mode and 0.7 for No-Thinking mode, respectively.

The additional results are present in 5. The qualitative trends are similar to content in 3.2.

D Implementation Details for Contrastive Representation Learning in Section 4.2

This appendix provides additional implementation details for the contrastive representation learning framework described in Section 4.2.

We use the MATH (Hendrycks et al., 2021) dataset as a training set, utilizing its training split



(a) *Distill-LLaMA-8B's tokens v.s. difficulty.*

(b) *Distill-LLaMA-8B's accuracy v.s. difficulty.*

Figure 5: Comparison of *Thinking* mode and four SeedThink variants using DeepSeek-R1-Distill-LLaMA-8B across different difficulty levels of the MATH500 dataset. The reported token counts include the tokens consumed by seed generation.

of 7,500 math problems with solutions. Each problem q_j in the dataset is first mapped to a fixed-dimensional feature vector $\mathbf{f}_j \in \mathbb{R}^D$. In our implementation, we adopt a sentence-level encoder Qwen3-Embedding-0.6B to obtain compact problem features. This encoder maps the raw problem text to a moderate-dimensional embedding with $D = 1024$, striking a balance between semantic expressiveness and computational efficiency. All problem features are precomputed and cached.

Given problem features $\mathbf{f}_j \in \mathbb{R}^D$, we introducing a lightweight projection $E_\theta : \mathbb{R}^D \to \mathbb{R}^d$ that allows the model to learn a task-specific notion of similarity that is explicitly shaped by utility-aware supervision. The reduced dimensionality further improves optimization efficiency and makes prototype geometry easier to control.

In the current framework, each seed-generation method $g_i \in \mathcal{G}$ is represented by a learnable prototype vector $\mathbf{e}_i \in \mathbb{R}^d$. This design keeps the selector extremely lightweight: the number of additional parameters grows linearly with the number of methods and is independent of problem count.

Training details. All embeddings are ℓ_2-normalized before similarity computation. We jointly optimize the encoder parameters θ and the method prototypes $\{\mathbf{e}_i\}$ using stochastic gradient descent with mini-batches sampled uniformly from the training set. Unless otherwise stated, we use the soft utility distillation objective in Eq. (9), which we found to be more stable than hard assignment when multiple seed methods exhibit comparable utilities.

E Supplementary materials for Experiment Setup

All experiments reported in this paper are conducted on a single machine with 8× NVIDIA A100 80GB GPU.

E.1 The Considered Seed Generation Methods $\mathcal{G}$

Beyond the seed types defined in Section C, our experiments expand the space of seed-generation strategies to better reflect the diversity of practical prompting behaviors and to strengthen the robustness of the selector. Rather than relying on a single notion of "good" seeds, we intentionally construct $\mathcal{G}$ to span multiple levels of reasoning granularity and computational investment, allowing us to study how different seed behaviors trade off solution quality and generation cost.

Type-A: Granularity-controlled seed generation. As the core dimension, we retain the four fine-grained seed types introduced in C

Type-B: Thinking-mode seed generation. Beyond granularity, we further consider how *computational effort during seed generation* affects end-to-end performance. Compared with the default *No-thinking* seed generator used in the pilot study, we introduce *Thinking*-mode variants that allow additional deliberation when producing the seed. For each granularity level in Type-A, we construct a paired Thinking-mode method that is identical in format and content constraints, differing only in whether the seed is generated with internal deliberation.

The motivation is that allocating more budget at the seed stage can sometimes lead to a higher-quality seed-thought that better constrains the sub-

sequent reasoning trajectory. Importantly, this does not necessarily increase the total inference cost. In practice, a stronger seed may reduce exploration, backtracking, and redundant computations in the final reasoning stage, thereby shortening the completion and lowering overall token usage.

Type-C: Exploration-based seed generation under No-thinking. In addition to Thinking-mode, we consider an alternative strategy for improving seed quality that avoids expensive internal deliberation. The key idea is to perform *broader exploration* under *No-thinking* mode, leveraging multiple or iterative lightweight generations instead of a single costly one. To keep token usage under control, we restrict Type-C methods to *high-level* seed types whose lengths are typically short (i.e., Outline and Key-steps seeds).

We study two exploration-based variants:

- *Multi-sample voting seed:* Multiple candidate seeds are independently generated under *No-thinking*, and the final seed is selected via majority voting over their structural or semantic consistency. The number of parallel generations is capped at five.
- *Iterative refinement seed:* A seed is generated under *No-thinking* and then iteratively refined in subsequent rounds, with each iteration taking the previous seed as input. The number of refinement steps is limited to five.

Compared with Thinking-mode, Type-C methods trade depth of deliberation for diversity of exploration. They allow the model to search for higher-quality seeds through breadth or incremental improvement while maintaining a bounded token budget. This design enables a controlled comparison between *deliberation-heavy* (Type-B) and *exploration-heavy* (Type-C) strategies for seed quality enhancement.

Final method set. The final candidate set $\mathcal{G}$ combines Type-A granularity-controlled seeds, their Thinking-mode counterparts (Type-B), and exploration-based No-thinking variants (Type-C). All methods are instantiated using fixed and reproducible prompt templates, with explicit limits on parallelism and iteration depth. This expanded method universe allows the selector to learn nuanced trade-offs between seed depth, deliberation cost, and exploration efficiency in a realistic and diverse prompting space.

Table 6: **Performance on the hard in-domain dataset AIME 2024.** The selector is trained on the MATH dataset and directly applied to AIME 2024 without additional training. Utility is computed by Eq. 3 with $\lambda = 0.5$.

Method	Accuracy↑	Speedup↑	Length↓	Utility↑
DeepSeek-R1-Distill-LLaMA-8B				
Think	46.67%	1.0×	11,288	-
NoThink	10.00%	4.9×	1,917	-
JointThink	43.33%	0.6×	15,220	0.217
CoThink	40.00%	1.3×	8,630	**0.318**
SeedThink (ours)	43.33%	2.3×	9,049	0.316
Qwen3-32B				
Think	75.00%	1.0×	12,571	-
NoThink	26.67%	4.9×	2,769	-
JointThink	75.00%	0.7×	19,337	0.375
CoThink	73.33%	1.4×	9,416	0.492
SeedThink (ours)	72.50%	2.8×	8,408	**0.528**

F Additional In-domain Transferability for Section 5.4.1 on AIME2024

To further evaluate SeedThink on harder mathematical reasoning problems, we report additional results on AIME 2024. Compared with GSM8K, AIME contains substantially more challenging competition-level problems, where direct *NoThink* generation suffers a much larger accuracy degradation. Therefore, AIME provides a useful stress test for examining whether SeedThink can still reduce reasoning cost without collapsing to shallow direct answering.

As shown in Table 6, the gap between *Think* and *NoThink* is much larger on AIME 2024 than on easier math benchmarks. For example, on Qwen3-32B, *NoThink* drops from 75.00% to 26.67%, indicating that direct answering is insufficient for these harder problems. SeedThink substantially narrows this gap: it achieves 72.50% accuracy, only 2.50 percentage points below *Think*, while providing a 2.8× runtime speedup and reducing the average length from 12,571 to 8,408 tokens. A similar pattern is observed on DeepSeek-R1-Distill-LLaMA-8B, where SeedThink matches JointThink in accuracy but is much faster, achieving a 2.3× speedup instead of slowing inference down. These results suggest that SeedThink remains effective in harder reasoning regimes where pure *NoThink* generation fails, and that seed-conditioned reasoning can preserve much of the benefit of deliberative thinking while reducing unnecessary computation.

G Additional Out-of-domain Evaluation for Section 5.4.2 on Scientific Reasoning

To further examine whether SeedThink generalizes beyond mathematical and financial reasoning, we conduct an additional evaluation on GPQA-Diamond, a challenging scientific reasoning benchmark designed for graduate-level expert questions. This setting differs substantially from mathematical reasoning: the problems often require specialized scientific knowledge, multi-step conceptual inference, and careful elimination among plausible answer choices. Therefore, GPQA-Diamond provides a useful testbed for evaluating whether SeedThink's efficiency gain is tied to math-specific prompting or can transfer to broader expert-level reasoning tasks.

Since GPQA-Diamond does not provide difficulty labels required by our adaptive seed selector, we use a fixed seed prompt, `detailed_steps`, for all test instances. This setting is intentionally conservative: it removes the benefit of input-adaptive seed selection and evaluates whether a single generic seed strategy can still provide useful reasoning guidance. As shown in Table 7, SeedThink consistently improves the efficiency–accuracy trade-off compared with the baselines. Compared with *Think*, SeedThink reduces the average generation length from 7,643 to 5,013 tokens and decreases the average runtime from 459.3s to 137.6s, corresponding to a $3.3\times$ speedup. Meanwhile, it preserves much of the accuracy advantage of deliberative reasoning, achieving 62.63% accuracy compared with 66.16% for *Think*. In contrast, *NoThink* is substantially faster but suffers a larger accuracy drop, while CoThink consumes many more tokens and runtime than SeedThink but achieves lower accuracy. These results suggest that SeedThink can provide effective reasoning acceleration on non-mathematical expert domains, including scientific reasoning.

H The Impact of the Trade-off Parameter λ

The trade-off parameter λ in Eq. (3) controls the relative importance of solution correctness and token saving when matching problems to seed-generation methods. A larger λ makes the selector more accuracy-oriented, while a smaller λ encourages more aggressive reduction in generation cost. Thus, λ provides a simple and interpretable knob for adapting SeedThink to different deployment preferences.

Table 7: Performance on complex scientific reasoning using GPQA-Diamond.

Method	Tokens↓	Runtime(s)↓	Acc.↑
Think	7,643	459.3	66.16%
NoThink	1,123	64.1	54.55%
CoThink	5,497	315.9	58.08%
SeedThink	5,013	137.6	62.63%

Table 8: **Sensitivity to the trade-off parameter λ.** Accuracy and average token counts under different λ values. Larger λ emphasizes correctness, while smaller λ encourages more aggressive token saving. Results are tested on Math500 using Qwen3-8B.

λ	0.1	0.3	0.5	0.7	0.9
Acc.↑	93.80%	94.00%	94.40%	95.20%	96.00%
Tokens↓	2,514	2,790	3,639	4,072	4,528

To study the impact of this parameter, we evaluate SeedThink under different λ values. As shown in Table 8, increasing λ leads to higher accuracy together with longer generations. Specifically, when λ increases from 0.1 to 0.9, accuracy improves from 93.80% to 96.00%, while the average token count increases from 2,514 to 4,528. Conversely, smaller λ values achieve stronger token reduction with a moderate accuracy drop. These results show that SeedThink offers a controllable accuracy–efficiency trade-off rather than relying on a fixed seed-selection policy.

I The Impact of the Seed length

To investigate whether longer or more abstract seeds lead to saturation in speculative gains, we conducted additional experiments. As shown in Table 9, rather than observing saturation, we find that **longer seeds generally increase speculative gains** (speedup from **1.47 to 1.64** and MAT from **1.65 to 1.96**), as they produce more opportunities for suffix matches. **We do not observe saturation within the studied range**, and we note that seed length is intentionally constrained: excessively long seeds would undermine the two-stage pipeline's efficiency. Within this realistic range, speculative gains (Mean accepted tokens) grow monotonically.

Table 9: Speculative gains (using SAM decoding) of different seed prompt Qwen3-14B on the Math500 dataset. MAT short for Mean Accept Token.

Seeds	MAT	Throughput (tokens/s)	Speed up from SD	Length	Accuracy
Think (w/o SAM)	0	28.44	–	4,741	96.8%
Think (w/ SAM)	1.65	40.26	× 1.42	4,776	96.8%
SeedThink (outline seed)	1.73	41.72	× 1.47	2,959	93.8%
SeedThink (key_steps seed)	1.82	43.88	× 1.54	2,847	93.4%
SeedThink (general_approach seed)	1.88	45.04	× 1.58	2,735	95.6%
SeedThink (detailed_steps seed)	1.96	46.87	× 1.64	2,650	94.4%

J Analysis of Reasoning Trajectory Optimization

We attribute a substantial portion of LRM inefficiency to two token-level patterns: *reflection* and *branching*. Reflection corresponds to repeated self-checking that prolongs generation after the solution direction is already determined, often marked by phrases such as "wait", "let me check", and "let me verify" (Wang et al., 2025a). Branching corresponds to exploring alternative solution paths signaled by cues such as "alternatively" or "another approach", which can be beneficial in principle but frequently leads to low-yield exploration in practice. We quantify both behaviors by counting reflection and branch tokens using the keyword rules described above.

Figure 4 shows that SeedThink consistently suppresses both patterns. On Qwen3-8B, reflection tokens decrease by 58.9% and branch tokens decrease by 72.6%. The effect strengthens with scale, and Qwen3-32B exhibits the largest reductions, with 69.5% fewer reflection tokens and 80.6% fewer branch tokens.

These reductions support a simple mechanism. The seed functions as a *reasoning anchor* that commits the model early to a coherent plan, which reduces uncertainty-driven verification loops and discourages costly detours into weak alternative branches. In other words, SeedThink improves efficiency by reshaping the reasoning trajectory, not merely by shortening outputs.

K Details on Prompt Templates

To avoid domain-specific prompt engineering, we use a unified set of seed-generation templates across datasets. Each template contains a domain placeholder, denoted as [domain], which is instantiated according to the benchmark under evaluation, e.g., "mathematical reasoning" for math benchmarks and "financial reasoning" for FinEval. Accordingly, domain-specific expressions such as formulas, financial concepts, accounting rules, or regulatory constraints are not hard-coded in the prompt, but are referred to through generic phrases such as "domain-relevant concepts, rules, evidence, or tools." This design allows the same prompting framework to be applied to different domains with only a lightweight substitution of the domain descriptor.

Outline seed generation prompt

```
<|User|>You are a reasoning assistant for
[domain] problems. Your job is to break
down a complex problem into high-level
reasoning steps. Focus only on outlining
the general approach or strategy. Do not
include specific numbers, domain-specific
formulas, detailed computations, or final
answers. Only describe the logic behind
solving the problem. Please break down
the following problem. [question]
<|Assistant|><think></think> \n
```

Key steps seed generation prompt

```
<|User|>You are a reasoning assistant for
[domain] problems. Your task is to rapidly
provide the key steps for solving the given
problem. Include the main reasoning steps
without performing detailed calculations
or giving the final answer. Mention any
important domain-relevant concepts, rules,
evidence, or tools that may be useful.
[question]
<|Assistant|><think></think> \n
```

General approach seed generation prompt

```
<|User|>You are a reasoning assistant
for [domain] problems. Quickly provide
a general approach to solve the given
```

```
problem. Include a step-by-step plan with
brief explanations of each step. You may
mention relevant domain concepts, rules,
evidence, or tools, but avoid extensive
calculations or overly specific details.
The goal is to set up the problem for
detailed reasoning later. [question]
<|Assistant|><think></think> \n
```

Detailed steps seed generation prompt

```
<|User|>You are a reasoning assistant
for [domain] problems.  Your task is
to promptly provide detailed steps for
solving the given problem.  Include
necessary setups, intermediate reasoning,
and relevant domain concepts, rules,
evidence, or tools. You may perform simple
calculations or checks if needed. However,
leave the most complex calculations or
reasoning for the next stage, and do not
give the final answer. [question]
<|Assistant|><think></think> \n
```

SeedThink generation prompt

```
<|User|>You are a reasoning assistant for
[domain] problems. Here is a seed solution
for reference: [Reference answer]
It is crucial to critically evaluate the
information provided in the reference,
recognizing that it may be incomplete,
biased, or incorrect.  If you think the
reference is incorrect, revise it using
your own reasoning and domain-relevant
knowledge.
[question]
<|Assistant|><think>\n
```