Thinking with Reasoning Skills: Fewer Tokens, More Accuracy

Guangxiang Zhao[1,¶], **Qilong Shi**[2,¶], **Xusen Xiao**[3], **Xiangzheng Zhang**[1,✉] **Tong Yang**[4], **Lin Sun**[1,✉]
[1]Qiyuan Tech, [2]Tsinghua University, [3]The University of Hong Kong, [4]Peking University
zhaoguangxiang@pku.edu.cn, stallone@pku.edu.cn, xiaoxusen@connect.hku.hk,
zhangxiangzheng@360.cn, yangtong@pku.edu.cn, sunlin1@360.cn
✉ **Correspondence:** zhangxiangzheng@360.cn, sunlin1@360.cn

Abstract

Reasoning LLMs often spend substantial tokens on long intermediate reasoning traces (e.g., chain-of-thought) when solving new problems. We propose to summarize and store reusable reasoning skills distilled from extensive deliberation and trial-and-error exploration, and to retrieve these skills at inference time to guide future reasoning. Unlike the prevailing *reasoning from scratch* paradigm, our approach first recalls relevant skills for each query, helping the model avoid redundant detours and focus on effective solution paths. We evaluate our method on coding and mathematical reasoning tasks, and find that it significantly reduces reasoning tokens while improving overall performance. The resulting lower per-request cost indicates strong practical and economic potential for real-world deployment.

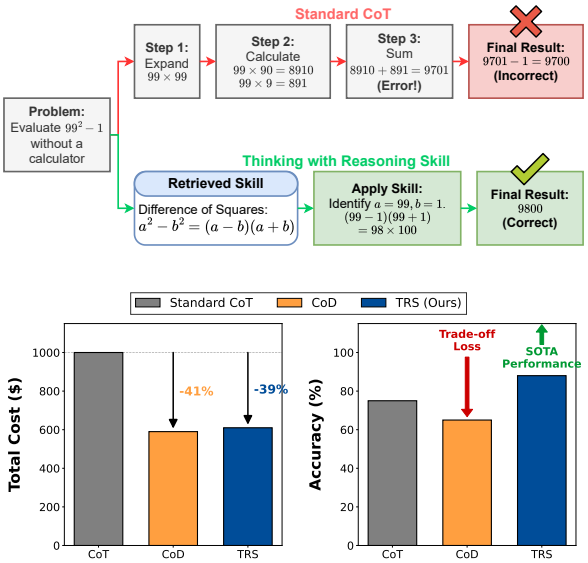


Figure 1: **Above:** The "Gist" of Thinking with Reasoning Skills. **Below:** Breaking the Efficiency-Accuracy Trade-off.

1 Introduction

Reasoning-centric large language models (LRMs) have rapidly evolved from research novelties to standard capabilities. Modern models, such as OpenAI o1 and DeepSeek-R1, now explicitly encourage intermediate deliberation to enhance reliability in math and code (OpenAI, 2024; DeepSeek-AI, 2025; Google, 2025; Anthropic, 2026; OpenAI, 2025). While this shift delivers impressive accuracy, it introduces a major bottleneck: *test-time compute is paid in tokens and latency*.

In practice, LRMs often generate thousands of "thinking" tokens, comprising redundant verification and trial-and-error loops (Han et al., 2024; Wang et al., 2025). Since commercial APIs bill by token—often pricing output higher than input (OpenAI, 2026)—lengthy traces dominate query costs and latency. Industry reports confirm that reasoning-heavy inference significantly amplifies infrastructure strain (Uptime Institute, 2025). Consequently, efficient reasoning is production-critical: we seek the benefits of deliberation without the cost of repeated rediscovery.

Recent work has proposed to shorten reasoning traces by controlling *how* models think. Prompt-only approaches compress intermediate traces (e.g., Chain-of-Draft) (Xu et al., 2025) or enforce explicit token budgets (e.g., TALE) (Han et al., 2024); decoding-time approaches suppress reflection markers to avoid overthinking loops (e.g., NoWait) (Wang et al., 2025). However, these methods still treat each query as a blank slate: the model is asked to re-derive solution logic from scratch, only faster. This often yields an efficiency–accuracy trade-off on hard problems: when the reasoning space is forcibly compressed, models may skip crucial steps and fail.

[*]Code: https://github.com/stallone0000/Reasoning-Skill. Dataset: https://huggingface.co/datasets/stallone0000/Reasoning-Skill. Demo: https://reasoning-skill.onrender.com/. The demo provides side-by-side comparisons of Direct and TRS reasoning. Latest demo URL will be updated in GitHub.

[¶]The first two authors contributed equally.

Humans solve this differently. Experts rarely re-derive everything; they recall reusable *skills* distilled from past problem solving (e.g., "look for an invariant", "use two-pointers", "apply chain rule"). Notably, *skill libraries and experience memory have become a mainstream design pattern in LLM agents*, enabling systems to reuse prior reflections or executable skills across tasks without weight updates (Yao et al., 2022; Shinn et al., 2023; Wang et al., 2023). Yet, for *pure reasoning* (math/coding) under token budgets, we still largely operate in a "reasoning from scratch" regime.

We propose **Thinking with Reasoning Skills (TRS)** (Figure 1): a training-free, retrieval-augmented framework that decouples *acquiring* reasoning logic from *executing* reasoning. Offline, we distill long deliberation trajectories (including trial-and-error) into compact, reusable *reasoning skills* with explicit triggers and pitfalls. Online, we retrieve the most relevant skills for a new query (even when no near-duplicate problems exist) and inject them into the prompt to steer the model toward an effective solution path. The key idea is that while questions are open-ended, *reasoning patterns are reusable*; we can replace redundant internal rediscovery with externalized procedural memory.

Empirically, TRS yields a surprising result: across both mathematical and coding benchmarks, it *reduces thinking tokens* while *improving accuracy* compared to standard reasoning from scratch, especially on harder subsets and for weaker models. This indicates that efficiency need not come at the expense of correctness—if we reuse distilled experience rather than merely forcing brevity.

Our main contributions are:

- **A practical framework for token-efficient reasoning:** TRS distills deliberation into retrievable *reasoning skills* and reuses them at inference time, remaining compatible with black-box LLM APIs.
- **Breaking the efficiency–accuracy trade-off:** on math and coding tasks, TRS consistently reduces thinking tokens/cost while matching or improving accuracy over strong baselines (e.g., CoD/TALE-style prompts).
- **Analysis for deployment:** we study transferability across models, output length, and retrieval ablations, showing TRS benefits grow on harder problems and can transfer across heterogeneous LLMs.

2 Related Work

Deliberative reasoning and test-time compute. Chain-of-Thought (CoT) prompting improves multi-step reasoning by eliciting intermediate traces (Wei et al., 2022), and subsequent work amplifies this effect via sampling/aggregation (self-consistency) (Wang et al., 2022) or structured search over thoughts (e.g., Tree-of-Thoughts) (Yao et al., 2023). Agent-style prompting (e.g., ReAct) further couples reasoning with actions/tools (Yao et al., 2022). While powerful, these paradigms often increase test-time compute, motivating the recent surge of *efficient reasoning* research.

Token-efficient reasoning via compression, budgets, decoding, or post-training. A first line of work reduces output length purely by prompting, e.g., compressing traces into minimalist drafts (Chain-of-Draft) (Xu et al., 2025), or explicitly constraining reasoning budgets (TALE) (Han et al., 2024). Decoding-time interventions suppress reflection markers correlated with redundant overthinking loops (NoWait) (Wang et al., 2025), but typically require logit-level control. Other approaches modify the model to internalize efficiency (e.g., post-training for budget awareness (Han et al., 2024)) or shift reasoning into alternative representations (e.g., latent-space reasoning in Coconut) (Hao et al., 2024). In parallel, recent work explores pruning/sparsifying reasoning computation for lower cost (e.g., reasoning-aware attention sparsity) (Hu et al., 2025). TRS is complementary: instead of only "thinking shorter" or changing decoding/training, we reduce detours by reusing distilled procedural experience.

Retrieval, memory, and reusable skills / templates. Retrieval augmentation has been studied beyond factual QA, showing gains even for non-knowledge-intensive tasks when retrieval is integrated properly (Guo et al., 2023; Chen et al., 2022). In agent systems, memory and skill libraries reuse past experience to improve future behavior without weight updates (e.g., Reflexion) (Shinn et al., 2023), and embodied agents can accumulate an explicit skill library for compositional reuse (e.g., Voyager) (Wang et al., 2023). Closest to our framing, recent "thought template" methods build reusable reasoning templates (BoT) (Yang et al., 2024) or distill templates into smaller models with self-correction (SuperCorrect) (Yang et al., 2025), and caching frameworks retrieve prior high-

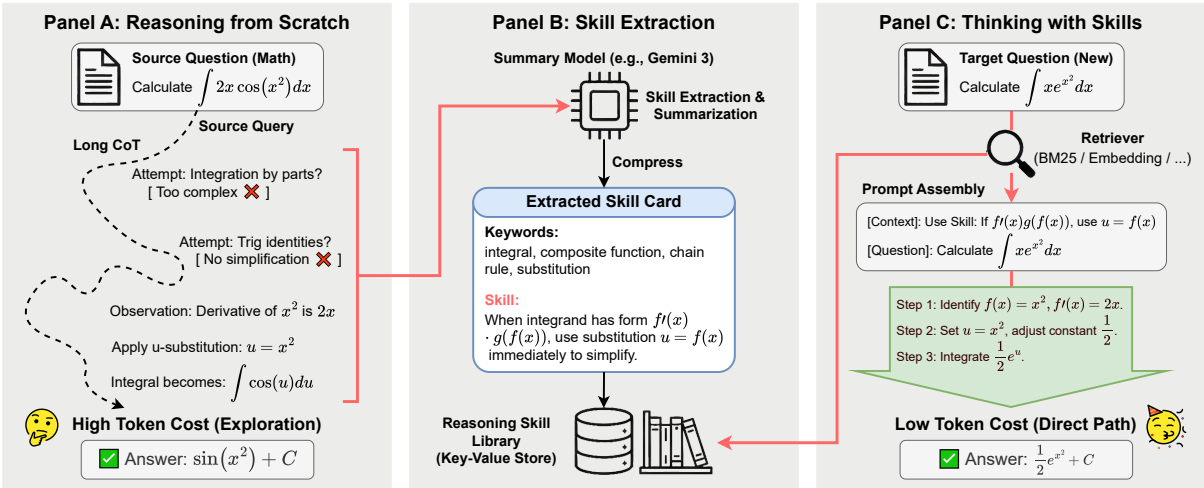


Figure 2: The process of Thinking with Reasoning Skills (TRS). **(A)** A standard reasoning model solves a source problem via a "reasoning from scratch" approach, involving redundant steps and trial-and-error (high token cost). **(B)** A summarizer model distills this long trajectory into a reusable, abstract reasoning skill (e.g., identifying the Chain Rule pattern), which is stored in a Key-Value library. **(C)** When a new, independent problem with similar underlying logic is presented, the relevant skill is retrieved and injected into the prompt. This guides the model to follow a direct solution path, significantly reducing reasoning tokens while maintaining high accuracy.

quality reasoning outputs to assist cheaper models (Cache-of-Thought) (Wu et al., 2025). TRS differs in goal and design: we target *token-efficient reasoning* for math/coding in a black-box-compatible pipeline, distilling both *successful strategies and failure-mode fixes* into compact *reasoning skills*, and we systematically study cross-model transfer, gating strength, skill length, and retrieval ablations under difficulty slicing.

3 Methodology

Goal and interface. We target *production-style* reasoning where cost and latency scale with test-time tokens. Given a query q (math or coding), a reasoning model $\mathcal{M}_r$ generates an intermediate trace τ and a final output y. Correctness is determined by $\mathrm{Eval}(y, a)$: exact match for math, and pass@1 against unit tests for code. Our goal is to *reduce reasoning tokens* (thinking length) while maintaining or improving accuracy. Unless the model exposes separate thinking tokens, we use completion tokens as a consistent proxy across methods.

3.1 Thinking with Reasoning Skills (TRS)

Figure 2 illustrates TRS. The key idea is to replace repeated "reasoning from scratch" with *retrieved procedural experience*. TRS is **training-free** and **black-box compatible**: offline we distill long trajectories into reusable skill cards; online we retrieve and inject a few cards to guide $\mathcal{M}_r$ toward a direct solution path. This decouples expensive exploration (one-time) from cheap reuse (per request).

3.2 Offline Skill Library Construction

For each source instance (q_i, a_i), we run $\mathcal{M}_r$ to obtain (τ_i, y_i) and compute $c_i = \mathrm{Eval}(y_i, a_i)$. We form an *experience record* $(q_i, a_i, \tau_i, y_i, c_i)$ and distill it with a stronger summarizer $\mathcal{M}_s$ into: (i) a compact *skill card* s_i and (ii) retrieval triggers K_i (10–20 keywords). If c_i=1, s_i captures the essential pattern (minimal transformation / invariant / algorithmic template). If c_i=0, s_i captures a reusable *failure-mode fix* (anti-pattern $\rightarrow$ correction). We standardize skill cards as short structured text: **Trigger / Do / Avoid / Check / Risk**. Extraction prompts, schema constraints, and card validation rules are in Appendix A.

Key–value storage. Each card becomes a key–value entry in a library $\mathcal{L} = \{(x_i \rightarrow v_i)\}$, where $v_i = s_i$ and $x_i = \mathrm{Concat}(q_i, K_i)$. Concatenating q_i with trigger keywords improves recall for sparse retrieval while keeping keys human-interpretable.

3.3 Online Retrieval and Skill Injection

Given a new query q, TRS retrieves top-k relevant skill cards and prepends them to the prompt of $\mathcal{M}_r$.

Retrieval backends. We support sparse BM25, dense retrieval with embeddings + nearest-neighbor search, and hybrid retrieval. We use **BM25** as de-

fault for math and **hybrid** for coding, following ablations (Appendix B).

Prompting and lightweight gating. Retrieved skills may be partially irrelevant or conflicting. We therefore use a lightweight arbitration instruction: *use only directly applicable skills; ignore irrelevant/contradictory advice*. To prevent prompt inflation, we (i) cap each card to a small budget (few bullets), and (ii) truncate to top-k by retrieval score. We analyze gating strength and skill length budgets in Section 6.

TRS prompt template

Retrieved Reasoning Skill(s).

1. **Skill** $s_{(1)}$**.** [A short, actionable skill distilled from prior trajectories.]
2. **Skill** $s_{(2)}$**.** [Optional; only if $k > 1$.]
3. ...

Instruction. Prefer the most directly applicable skill; ignore irrelevant or contradictory advice. Keep intermediate reasoning concise while maintaining correctness.

Task. Solve the following question. Use the retrieved skills above when relevant.

Question. [Insert the new query q here.]

Figure 3: A standard TRS prompt template that injects retrieved reasoning skills before the query.

Why this reduces tokens in practice. TRS reduces reasoning tokens by replacing redundant exploration (branching, detours, repeated debugging) with a retrieved procedural shortcut and explicit pitfalls distilled from failures. Although TRS adds a small input prefix, it typically yields a larger reduction in generated reasoning tokens, leading to lower end-to-end cost and latency; we quantify token/cost trade-offs in our experiments.

4 Experimental Setup

Math benchmark (disjoint split). We evaluate on DEEPMATH-103K with a strict split: **93K** source problems for library construction and **10K** held-out problems for evaluation. Unless stated otherwise, we use BM25 retrieval with k=1 on math, which is consistently best in our analysis (Appendix B).

Coding benchmark (end-to-end judging). We evaluate on NEMOTRON-COMPETITIVE-PROGRAMMING-V1 (34K). We filter to **26.6K** instances with reliable local judging signals and hold out **1K** for testing. We generate solution trajectories and verdicts using two generators (Gemini 3 Flash Preview and Doubao Seed 2.0 Pro), distill structured experience cards with Gemini 3 Flash Preview, and build a dual-route retrieval index (BM25 + dense). At inference time, we retrieve **top-5** cards and inject them into the prompt; the model receives only (cards + problem) with no additional reasoning-style prompts. Final correctness is determined by a local compile-and-run judge on test cases (pass@1). Full pipeline, filtering rules, outcome partitions, and card statistics are in Appendix C and Appendix D.

Models and summarizer. For inference-time evaluation, we compare TRS against strong baselines on multiple models, including **GPT-OSS-120B**, **Gemini-3 Flash Preview**, and **GPT-4o-mini**. We also evaluate on the **Doubao Seed** family: we use **Seed-1.8 Pro** for math and **Seed-2.0 Pro** for coding, since the coding experiments were conducted later and thus adopted the latest available Seed model. For skill extraction across all experiments, we use a fixed strong summarizer (Gemini Flash) and fixed distillation prompts (Appendix A). For dense retrieval, we use BGE-M3 embeddings with FAISS (Appendix B).

4.1 Metrics

We report: **Accuracy** (exact match for math; pass@1 for coding), **Reasoning tokens** (think tokens when exposed; otherwise output length proxy), and **Cost reduction** relative to the baseline under a representative input/output token pricing ratio. Token accounting details are provided in Appendix F.

4.2 Difficulty Slices by Baseline Thinking Length

To isolate hard instances, we slice the test set by the baseline model's thinking length. Let $T_{\text{think}}^{\text{base}}(q)$ be baseline think tokens for query q. For threshold θ, define $\mathcal{Q}_{\geq\theta} = \{q : T_{\text{think}}^{\text{base}}(q) \geq \theta\}$. Larger θ indicates harder instances (baseline "thinks longer"). We report accuracy and token/cost trends across thresholds to quantify where TRS yields the largest benefits.

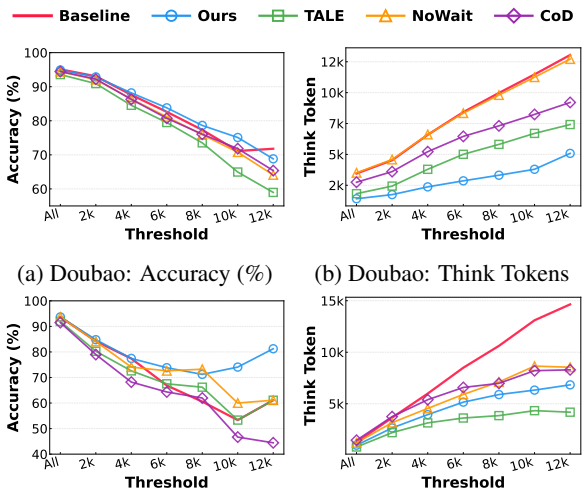
5 Main Results

Table 1 shows that TRS improves the accuracy–efficiency trade-off on both math and coding. On

Model	Method	Math Acc.	Math Token #	Math Cost	Coding Acc.	Coding Token #	Coding Cost
Gemini-3-Flash	Direct	94.8%	7490	100.0%	72.0%	20,206	100.0%
	TRS	95.5% ↑0.7%	6106 ↓18.5%	82.5% ↓17.5%	71.7% ↓0.3%	17,072 ↓15.5%	85.2% ↓14.8%
Doubao Seed†	Direct	95.1%	3453	100.0%	63.6%	7,500	100.0%
	TRS	94.9% ↓0.2%	1411 ↓59.1%	46.2% ↓53.8%	64.4% ↑0.8%	6,730 ↓10.3%	94.0% ↓6.0%
GPT-OSS-120B	Direct	93.7%	1257	100.0%	54.2%	5,080	100.0%
	TRS	93.7% →0%	976 ↓22.4%	83.1% ↓16.9%	58.3% ↑4.1%	5,177 ↑1.9%	104.8% ↑4.8%
GPT-4o-mini	Direct	59.6%	819	100.0%	22.0%	726	100.0%
	TRS	61.4% ↑1.8%	650 ↓20.6%	93.1% ↓6.9%	24.4% ↑2.4%	480 ↓33.9%	93.7% ↓6.3%

Table 1: Comparison of DIRECT and TRS on math and coding datasets. Token # denotes average *completion* tokens. Cost is normalized to Direct = 100.0% within each model–dataset pair under a fixed input/output token price ratio. †We use different Doubao Seed versions: Seed-1.8 Pro for math and Seed-2.0 Pro for coding. Cyan indicates improvement.

math, TRS reduces completion tokens and cost while preserving or improving accuracy: Gemini-3-Flash improves by +0.7 with a 17.5% cost reduction, GPT-4o-mini improves by +1.8 with lower cost, Doubao Seed cuts cost by 53.8% with only a -0.2 change, and GPT-OSS-120B preserves accuracy while reducing cost by 16.9%. On **coding**, TRS generally improves pass@1 with lower cost when measured: GPT-4o-mini increases to 24.4% (+2.4) with a 6.3% cost reduction, Doubao Seed improves by +0.8 with a 6.0% cost reduction, and GPT-OSS-120B improves by +4.1 though cost slightly increases (+4.8%) due to larger prompts. Overall, TRS gains efficiency by *reusing distilled procedural skills* instead of forcing shorter reasoning, often reducing per-request cost without sacrificing correctness.



(a) Doubao: Accuracy (%) (b) Doubao: Think Tokens

(c) OSS: Accuracy (%) (d) OSS: Think Tokens

Figure 4: Performance comparison across difficulty thresholds (Doubao vs. OSS).

6 Analysis

6.1 Comparison with Baselines

We compare **TRS** against **TALE-EP** (budget-constrained), **Chain of Draft (CoD)** (brevity-constrained), and **NoWait**. Figure 4 details performance sliced by baseline think token count θ.

Breaking the Trade-off. Existing methods suffer severe accuracy trade-offs on hard problems. As difficulty θ increases, TALE and CoD show catastrophic collapse (e.g., dropping below baseline on GPT-OSS for $\theta > 10$k), suggesting enforced brevity cripples deep reasoning. In contrast, **TRS** consistently matches or exceeds baseline accuracy, achieving significant uplift (∼45% to ∼80%) on the hardest GPT-OSS tasks. Retrieved skills act as a "navigation map," preventing the model from getting lost in incorrect branches.

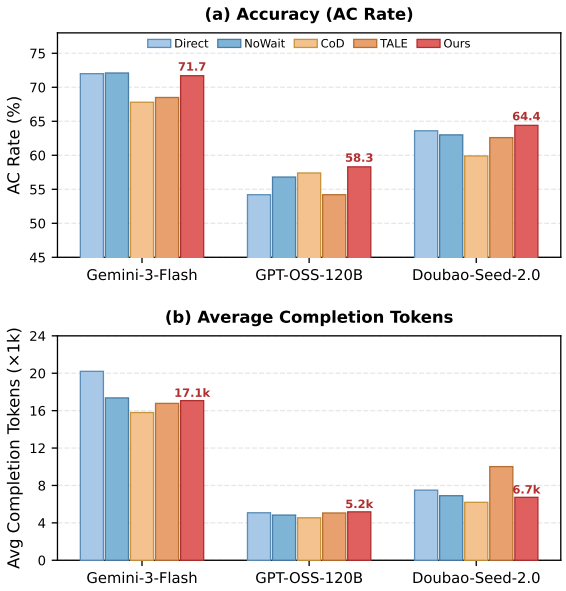


Figure 5: Compare to Direct on coding competitions.

Analysis	Condition	Doubao	OSS
Representation	Direct	92.8	90.6
	Raw examples	92.0	87.2
	Raw CoT	91.0	89.8
	Free summary	91.6	86.2
	Structured card (2.5k)	91.2	85.2
	Full TRS (93k)	92.6	91.2
Coverage	10k structured cards	91.8	89.2
	30k structured cards	90.6	87.4
	60k structured cards	92.2	90.8
	93k structured cards	92.6	91.2

Table 2: Controlled representation and coverage analysis on a fixed 500-question DeepMath subset. Values are accuracy percentages; only the retrieved representation or structured-library size changes.

Model	Acc. Δ	Out Tok. Δ	Cost Δ
Doubao-1.8	+1.88	↓4.9%	↓2.8%
Doubao-2.0-Pro	-1.11	↓17.5%	↓15.7%
Gemini-3-Flash	+0.92	↑8.6%	↑9.0%
GPT-OSS-120B	+0.50	↓8.1%	↓6.5%
GPT-OSS-20B	-0.60	↓5.3%	↓4.3%

Table 3: Average TRS-minus-direct deltas on the 120-question external contest-math suite using the pure 7,616-card AoPS library. Accuracy is reported in percentage points; output-token and cost changes are relative to Direct, following the accounting style of Table 1. Because direct and TRS use 32 and 8 repeats per question, respectively, we treat this comparison as descriptive rather than a formal significance test.

Efficiency Gains. On Doubao, TRS achieves the **lowest** token consumption, cutting generated tokens for hard problems ($\theta \geq 12$k) from $\sim$12k to $\sim$5k while maintaining top accuracy. On GPT-OSS, TRS strikes the optimal balance: it halves token usage ($\sim$15k to $\sim$7k) without the accuracy loss seen in "speed-limit" approaches like TALE or CoD.

6.2 Controlled Representation and Coverage Analysis

To isolate the effect of *structured reasoning-skill distillation* from simple retrieval, we ran a same-setup control on a fixed 500-question DeepMath subset, keeping the evaluation subset, target model, prompt family, verifier, retriever, and top-k fixed while varying only the retrieved representation. Table 2 shows that naive retrieval does not explain the TRS gains: raw examples, raw CoT traces, free-form summaries, and small structured-card libraries all underperform full TRS, especially on OSS. The strongest result appears only when structured skill reuse is paired with sufficient library coverage, indicating that TRS is not merely prepending relevant-looking context but retrieving reusable procedural guidance from a broad skill bank.

6.3 Transfer to External Contest-Math Benchmarks with AoPS-Derived Skills

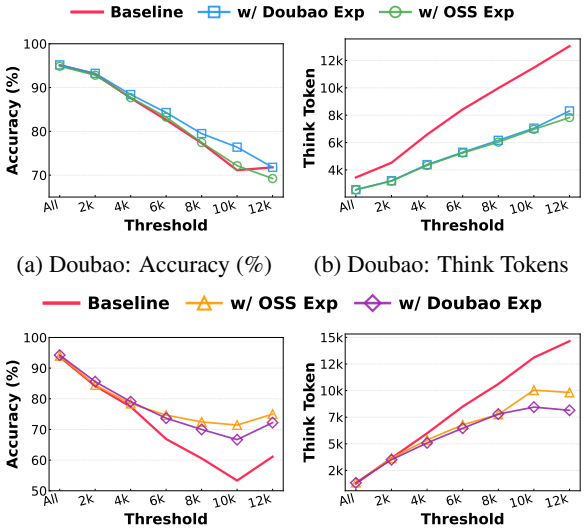
To test whether skills distilled from a separate contest-math corpus transfer beyond DeepMath, we built an external AoPS-derived library from 7,616 contest-math problems and evaluated TRS on 120 benchmark questions spanning AIME 2024 I, AIME 2024 II, AIME 2025, AIME 2026, and HMMT November 2025. In this study, direct prompting uses 32 repeats per question and TRS uses 8 repeats per question; TRS retrieves the top-1 BM25 match from the AoPS library and injects only the retrieved heuristic into a unified prompt. Table 3 shows a heterogeneous but meaningful transfer pattern: 13 of 25 model-benchmark pairs improve accuracy and 20 of 25 reduce per-query cost. The clearest average beneficiary is Doubao-1.8 (+1.88 accuracy points with a 2.8% cost reduction), while GPT-OSS-120B shows smaller but still positive average gains (+0.50) with a 6.5% cost reduction. By contrast, Gemini-3-Flash improves accuracy (+0.92) but increases output tokens and cost, and Doubao-2.0-Pro mainly trades a small amount of accuracy for a 15.7% cost reduction. We therefore treat external skill transfer as evidence of reusable reasoning structure in nearby domains, but not as a universal improvement guarantee. Appendix I provides the full 25-condition accuracy and cost-percentage heatmaps and setup details.

6.4 Cross-Model Skill Transfer

We investigate whether skills distilled from one model (Source) can transfer effectively to another (Target) using Doubao- and OSS-based libraries. Results in Figure 6 show:

- **Impact on Hard Problems:** The gains from TRS grow with problem difficulty θ. As baseline accuracy declines, TRS better preserves performance while further reducing cost, suggesting that retrieved skills effectively prune redundant exploration.

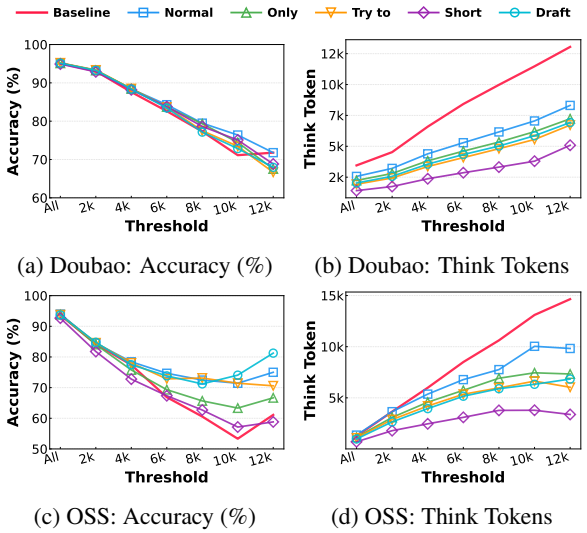
- **Source Alignment:** Skills are highly transferable across models. However, *same-source* skills (e.g., a Doubao library used for Doubao) generally yield the highest accuracy, likely due to stylistic alignment. Interestingly, *cross-*



(a) Doubao: Accuracy (%) (b) Doubao: Think Tokens

(c) OSS: Accuracy (%) (d) OSS: Think Tokens

Figure 6: Cross-model transfer: Skills from Doubao vs. OSS applied to target models.



(a) Doubao: Accuracy (%) (b) Doubao: Think Tokens

(c) OSS: Accuracy (%) (d) OSS: Think Tokens

Figure 7: Impact of skill-injection prompts on accuracy and efficiency.

source skills can sometimes lead to more aggressive token reduction.

- **Implication:** These results support a modular workflow: skills distilled from strong proprietary models can be used to improve more efficient deployment models, while model-specific distillation remains preferable when maximizing accuracy is the priority.

6.5 Prompt Strategy for Skill Injection

We ablate five prompt strategies (Normal, Only, Try to, Short, and Draft) to study the trade-off between accuracy and response length (Figure 7).

- **Doubao:** The **Short** prompt, which imposes an explicit budget, provides the best trade-off, achieving substantial token reduction while maintaining competitive accuracy. We therefore adopt it as the default for Doubao.
- **OSS:** The **Draft** prompt, which encourages a concise step-by-step style, achieves the highest accuracy while retaining strong efficiency. In contrast, the **Short** prompt leads to noticeably larger accuracy degradation. We therefore use **Draft** as the default for OSS.

Exact templates are provided in Appendix G.

7 Summarizer-Choice Ablation

As an additional check beyond the representation controls in Table 2, we regenerated the free-summary library with GPT-5-mini on the same fixed 500-question DeepMath subset. This variant reaches 89.4% on Doubao and 89.6% on OSS. Compared with the original free-summary row in Table 2, the alternative summarizer improves OSS but weakens Doubao, and it still does not match full TRS on OSS. The result suggests that summarizer choice matters, but the final TRS effect is not reducible to a single free-form summary backend.

8 Conclusion

In this paper, we introduced **Thinking with Reasoning Skills (TRS)**, a framework that addresses the high inference cost of reasoning models by shifting the paradigm from "reasoning from scratch" to "reasoning with recalled experience." Instead of forcing models to be brief—which often harms performance on complex tasks—TRS retrieves distilled, actionable *skill cards* (encompassing both successful strategies and failure-mode fixes) to guide the model through the solution space. Empirical results on large-scale mathematical and coding benchmarks demonstrate that TRS successfully breaks the efficiency–accuracy trade-off: it significantly reduces reasoning tokens and overall inference costs while matching or exceeding the accuracy of standard Chain-of-Thought baselines, particularly on harder problems. As a training-free, black-box compatible approach that generalizes across models, TRS offers a practical and scalable pathway for deploying capability-intensive reasoning models in real-world applications.

Limitations

First, our analysis of *failure cases* is not yet as deep as our success-skill study. While TRS can distill "failure-mode fixes" from incorrect trajectories, our current pipeline relies on limited supervision signals (e.g., final answer correctness for math, and pass/fail information from test cases for coding). We do not yet fully characterize which types of errors are most recoverable via skill distillation, nor do we have a principled method to summarize fine-grained failure dynamics beyond these signals.

Second, TRS introduces an additional retrieval and prompt-injection layer. Although it is lightweight compared to long test-time reasoning, the end-to-end benefit depends on (i) retrieval quality and (ii) controlling prompt inflation. In some settings (e.g., when completion tokens do not decrease or prompts grow too long), cost may not strictly improve even if accuracy increases.

Third, our current evaluation is still bounded. We focus mainly on math and competitive programming with automatic verifiers, plus targeted transfer and stress tests. Broader domains, weakly verifiable tasks, and open-ended workflows remain important future settings.

Finally, TRS opens a large design space. Future work can study which model should generate the original CoT, how to choose the summarizer, how to design the summarization and skill-use prompts, how to extend skill reuse to more domains, how to compress many problem-level skills into fewer general skills, and how to retrieve the most suitable skill from a very large library. The last question is especially connected to retrieval-augmented generation: as the skill library grows, retrieval design will become central to making experience reuse reliable and scalable.

Ethical Considerations

This work adheres to ACL's ethical guidelines, and we state that there are no ethical concerns to our knowledge.

Acknowledgements

We thank the anonymous reviewers, ACs, SACs, and PCs for their contributions.

References

Anthropic. 2026. Introducing claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6. Accessed: 2026-02-15.

Xiang Chen, Lei Li, Ningyu Zhang, Xiaozhuan Liang, Shumin Deng, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. 2022. Retrieval-augmented prompt learning. *arXiv preprint arXiv:2205.14704*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Google. 2025. A new era of intelligence with gemini 3. https://blog.google/products-and-platforms/products/gemini/gemini-3/. Accessed: 2026-02-15.

Zishan Guo, Hongyu Jin, Hongshen Tan, Xin Jiang, and Yuan-Fang Wang. 2023. Prompt-guided retrieval augmentation for non-knowledge-intensive tasks. In *Findings of the Association for Computational Linguistics: ACL 2023*.

Tingxu Han, Chunrong Fang, Shiyu Zhao, Shiqing Ma, Zhenyu Chen, and Zhenting Wang. 2024. Token-budget-aware LLM reasoning. *Preprint*, arXiv:2412.18547.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Coconut: Reasoning in continuous latent space for efficient inference. *arXiv preprint arXiv:2412.06769*.

Junhao Hu, Wenrui Huang, Weidong Wang, Zhenwen Li, Tiancheng Hu, Zhixia Liu, Xusheng Chen, Tao Xie, and Yizhou Shan. 2025. Raas: Reasoning-aware attention sparsity for efficient llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2577–2590, Vienna, Austria. Association for Computational Linguistics.

OpenAI. 2024. Openai o1 system card. https://openai.com/index/openai-o1-system-card/. Accessed: 2026-02-15.

OpenAI. 2025. Gpt-5 system card. https://openai.com/index/gpt-5-system-card/. Accessed: 2026-02-15.

OpenAI. 2026. Openai api pricing. https://openai.com/api/pricing/. Accessed: 2026-02-15.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*.

Uptime Institute. 2025. Ai reasoning will take a toll on infrastructure footprint. https://uptimeinstitute.com/blog/ai-reasoning-will-take-a-toll-on-infrastructure-footprint. Accessed: 2026-02-15.

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

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

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. *arXiv preprint arXiv:2201.11903*.

Mingyuan Wu, Jize Jiang, Haozhen Zheng, Meitang Li, Zhaoheng Li, Beitong Tian, Bo Chen, Yongjoo Park, Minjia Zhang, ChengXiang Zhai, and Klara Nahrstedt. 2025. Cache-of-thought: Master-apprentice framework for cost-effective vision language model reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 1895–1909, Suzhou, China. Association for Computational Linguistics.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025. Chain of draft: Thinking faster by writing less. *Preprint*, arXiv:2502.18600.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. 2024. Buffer of thoughts: Thought-augmented reasoning with large language models. *arXiv preprint arXiv:2406.04271*.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Minkai Xu, Joseph E. Gonzalez, Bin Cui, and Shuicheng Yan. 2025. Supercorrect: Advancing small llm reasoning with thought template distillation and self-correction. In *International Conference on Learning Representations (ICLR)*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

A Skill Card Schema, Validation, and Distillation

Schema. Each skill is a compact structured card with five fields: **Trigger** (applicability cues), **Do** (minimal actionable recipe), **Avoid** (anti-patterns), **Check** (must-verify constraints/invariants), **Risk** (edge cases/failure modes). We use the same schema for math/coding and for both successes and failures.

Distillation input and rule. For each experience record (q, a, τ, y, c), the summarizer produces a card s and 10–20 retrieval triggers K. If $c{=}1$, s summarizes the essential solution pattern. If $c{=}0$, s summarizes an anti-pattern and a concrete fix that generalizes beyond the instance.

Abstraction and safety constraints. We enforce three constraints: (i) **abstraction**: avoid copying instance-specific constants, answers, or full code; (ii) **actionability**: procedures must be executable as steps/checks; (iii) **compactness**: each field is short (few bullets). Cards violating schema or copying instance-specific details are discarded. We also discard cards with missing fields or malformed structure. The released code package includes the distillation, direct-inference, TRS-inference, and coding-specific prompt templates used to reproduce these steps.

B Retrieval Backends and Hybrid Fusion

Index keys. We index each card with $x = \mathrm{Concat}(q, K)$, where K are summarizer-generated triggers.

Sparse (BM25). BM25 over $\{x\}$ provides strong lexical matching when problems share surface triggers (common in math).

Dense (BGE-M3 + FAISS). We embed queries and keys with BGE-M3 and run FAISS nearest-neighbor search. Dense retrieval improves semantic matching when lexical overlap is weak (common in coding).

Hybrid retrieval. For coding, we use dual-route retrieval: BM25 candidates + dense candidates, merged with a simple score fusion and then truncated to top-k. We ablate BM25-only / dense-only / hybrid and k in Section 6. Default choices: math uses BM25 ($k{=}1$); coding uses hybrid ($k{=}5$).

Table 4: Comparison of retrieval strategies across mathematical reasoning and competitive programming. Math: 1,000 problems, Doubao-Seed-2.0 as generator, GPT-5-mini as verifier. Code: 1,000 holdout problems, Gemini-3-Flash as the inference model. "Δ" denotes the difference against the no-retrieval baseline.

Domain	Retrieval	Accuracy (%)	Δ (%)	Avg Output Tok.
Math	None (Baseline)	95.9	–	2,240
	BM25	95.5	−0.4	2,240
	Embedding	94.8	−1.1	2,268
	Hybrid	95.2	−0.7	2,311
Code	None (Baseline)	72.0	–	20,206
	BM25	71.5	−0.5	16,633
	Hybrid	72.0	±0.0	16,491

C Coding Pipeline: Data Filtering, Partitions, and Card Statistics

Dataset and split. We use NEMOTRON-COMPETITIVE-PROGRAMMING-V1 (34K). We filter to 26,641 training instances with reliable local judging signals and hold out 1,000 additional instances for evaluation. The 1,000-problem holdout is stratified into both_ac (633), flash_ac_doubao_wrong (95), flash_wrong_doubao_ac (21), and both_wrong (251).

Step 1: Data preparation and filtering. We retain instances that are consistently judged by our local compile-and-run evaluator and exclude instances with unstable verdicts (e.g., ambiguous I/O, inconsistent tests, or frequent runtime issues). This filtering reduces noise in both card distillation and final evaluation.

Step 2: Outcome partitioning (four types). We run two generators (Gemini 3 Flash Preview; Doubao Seed 2.0 Pro) and categorize each training instance by verdict: both_ac (16,851), flash_ac_doubao_wrong (2,521), flash_wrong_doubao_ac (570), both_wrong (6,699). These categories produce success experience, contrastive paired experience, and failure experience.

Step 3: Experience distillation (cards). We distill structured cards from trajectories using Gemini Flash. The v5 coding library released with the camera-ready artifacts contains 42,564 compact cards: success (31,044), contrast (5,069), and failure diagnostics (6,451), where the failure subset consists of wrong-approach cards (3,733) and edge-fix cards (2,718). To reduce redundancy and prompt inflation at inference time, we keep success cards from only one generator model, while preserving contrast/failure cards.

Step 4: Index construction. We build a dual-route retrieval index: BM25 over text keys and FAISS over dense embeddings (BGE-M3).

Step 5: Inference injection and evaluation. On the 1K test split, we retrieve top-5 cards and inject them into the prompt. The model receives only (retrieved cards + problem statement), with no additional reasoning-style prompting. Correctness is determined by local compile-and-run on test cases (pass@1).

D Local Coding Judge and Execution Environment

Judge protocol. For each generated program, we compile and run it against the provided test cases and compare outputs. A solution is correct iff it passes all tests (pass@1).

Environment. We use a fixed, sandboxed environment for all methods to ensure fairness. The released judge extracts Python/C++ code, compiles C++ with `g++ -O2 -std=gnu++17 -pipe`, runs Python with the local Python interpreter, applies a default 2-second timeout and 1024MB memory cap unless a problem specifies its own limits, and terminates non-terminating processes. The public release should be run inside an isolated sandbox because executing generated programs is inherently unsafe.

E BM25 vs Embedding

Table 4 shows the result.

Mathematical Reasoning. On the math benchmark, all three retrieval methods cause a slight accuracy drop compared to the no-retrieval baseline (95.9%). BM25 incurs the smallest degradation (-0.4%), whereas dense embedding retrieval suffers the most (-1.1%) and hybrid falls in between

(-0.7%). We attribute this to the fact that mathematical heuristics are best matched via lexical cues: BM25 excels at surface-level matching of formula patterns and theorem names, while embedding models—trained predominantly on natural language—tend to retrieve semantically plausible but structurally mismatched hints that occasionally mislead the generator. Meanwhile, token overhead is nearly identical across methods ($\sim$2,240–2,311 output tokens per problem). Given its best accuracy and lowest cost, we adopt pure BM25 for math retrieval.

Competitive Programming. On the coding benchmark, the pattern reverses: hybrid retrieval fully preserves baseline accuracy (72.0%), while BM25 alone shows a minor drop (-0.5%). This is expected because algorithmic problems benefit from semantic similarity—problems with different surface descriptions may share underlying algorithmic patterns that dense embeddings capture but keyword matching misses. Both retrieval strategies substantially reduce output tokens ($\sim$16,500 vs. 20,200), indicating that injected experience hints help the model generate more concise solutions.

F Token and Cost Accounting

Tokens. When explicit think tokens are exposed, we report thinking length directly. Otherwise, we report output token length as a proxy, applied consistently across compared methods.

Cost proxy. We compute cost from input and output token counts under a fixed input/output token price ratio for each model, but report only normalized percentages in the paper. Direct is set to 100.0% within each model–dataset pair, and TRS cost is reported relative to that baseline. A value below 100.0% indicates lower estimated per-query cost; deltas such as $\downarrow 17.5\%$ denote percentage reduction against Direct. This normalization keeps cost comparisons readable while preserving the effect of TRS's longer prompt and shorter generated reasoning.

G Prompt Templates

We provide the exact prompt templates used in our experiments to facilitate reproducibility. These templates illustrate how reasoning skills are injected into the model context and how baseline constraints are enforced.

TRS prompt: Normal

```
You are a helpful and harmless assistant.
You may be given an optional Solving Hints
section. Use it only if it is relevant to the
problem; otherwise, ignore it completely.
[Solving Hints] SOLVING_HINTS [/Solving Hints]
Problem: PROBLEM
```

Figure 8: Full prompt template for TRS-Normal.

TRS prompt: Only

```
You are a helpful and harmless assistant.
You may be given an optional Solving Hints
section. Use it only if it is relevant to the
problem; otherwise, ignore it completely.
[Solving Hints] SOLVING_HINTS [/Solving Hints]
Only try to reduce the number of tokens used
if the solution hints are useful; otherwise,
please think normally. Problem: PROBLEM
```

Figure 9: Full prompt template for TRS-Only.

TRS prompt: Try-to

```
You are a helpful and harmless assistant.
You may be given an optional Solving Hints
section. Use it only if it is relevant to the
problem; otherwise, ignore it completely.
[Solving Hints] SOLVING_HINTS [/Solving Hints]
If you use the solving hints, please try to
reduce the number of tokens used.  Problem:
PROBLEM
```

Figure 10: Full prompt template for TRS-Try-to.

TRS prompt: Short (budgeted)

```
You are a helpful and harmless assistant.
You may be given an optional Solving Hints
section. Use it only if it is relevant to the
problem; otherwise, ignore it completely.
[Solving Hints] SOLVING_HINTS [/Solving Hints]
Let's think step by step and use less than
[budget] tokens: PROBLEM
```

Figure 11: Full prompt template for TRS-Short (budgeted).

TRS prompt: Draft (CoD-like)

```
You are a helpful and harmless assistant.
You may be given an optional Solving Hints
section. Use it only if it is relevant to the
problem; otherwise, ignore it completely.
[Solving Hints] SOLVING_HINTS [/Solving Hints]
Think step by step, but only keep a minimum
draft for each thinking step, with 5 words at
most. Problem: PROBLEM
```

Figure 12: Full prompt template for TRS-Draft (CoD-like).

G.1 TRS Skill-Injected Prompts

Figures 8 through 12 present the different TRS skill-injection prompt variants evaluated in our ablation study (Section 6.5). These templates reflect different ways of incorporating retrieved skills into the model prompt, ranging from more permissive guidance to more explicit constraints on reasoning style and length.

We find that different models respond differently to such prompt constraints. Accordingly, based on the empirical results reported in Section 6.5, we adopt different default templates for different model families:

- For the **Doubao** family, we adopt the **Short** template (Figure 11) as the default, since it explicitly enforces a tighter budget and achieves the best efficiency–accuracy trade-off in our experiments.
- For **GPT-OSS-120B**, we select the **Draft** template (Figure 12) as the default, as it encourages a concise chain-of-draft style that better preserves accuracy while still reducing response length effectively.

G.2 Baseline Prompts

Figures 13, 14, and 15 display the prompts used for the comparison methods. These baselines rely on prompt-engineering techniques to constrain reasoning length (TALE-EP, CoD) or suppress specific tokens (No-Wait) without the aid of external retrieved skills.

TALE-EP prompt: Budget + Solve

```
Question:
{QUESTION}
Let's think step by step and use less than
[Budget Here] tokens:
--
Question:
{QUESTION}
Task: Analyze the given question and estimate
the minimum number of tokens required to
generate a complete and accurate response.
Please give the response by strictly following
this format: [[budget]], for example, Budget:
[[12]].
```

Figure 13: Full prompt template for TALE-EP (two-phase: budget estimation and solve).

CoD prompt

```
Question:
{QUESTION}
Think step by step, but only keep a minimum
draft for each thinking step, with 5 words at
most. Return the answer at the end of the
response after a separator ####.
```

Figure 14: Full prompt template for CoD.

No-Wait prompt

```
Question:
{QUESTION}
Think step by step. Do not use any of the
following words in your thinking process:
"wait", "alternatively", "hmm", "but",
"however", "alternative", "another", "check",
"double-check", "oh", "maybe", "verify",
"other", "again", "now", "ah", "any".
```

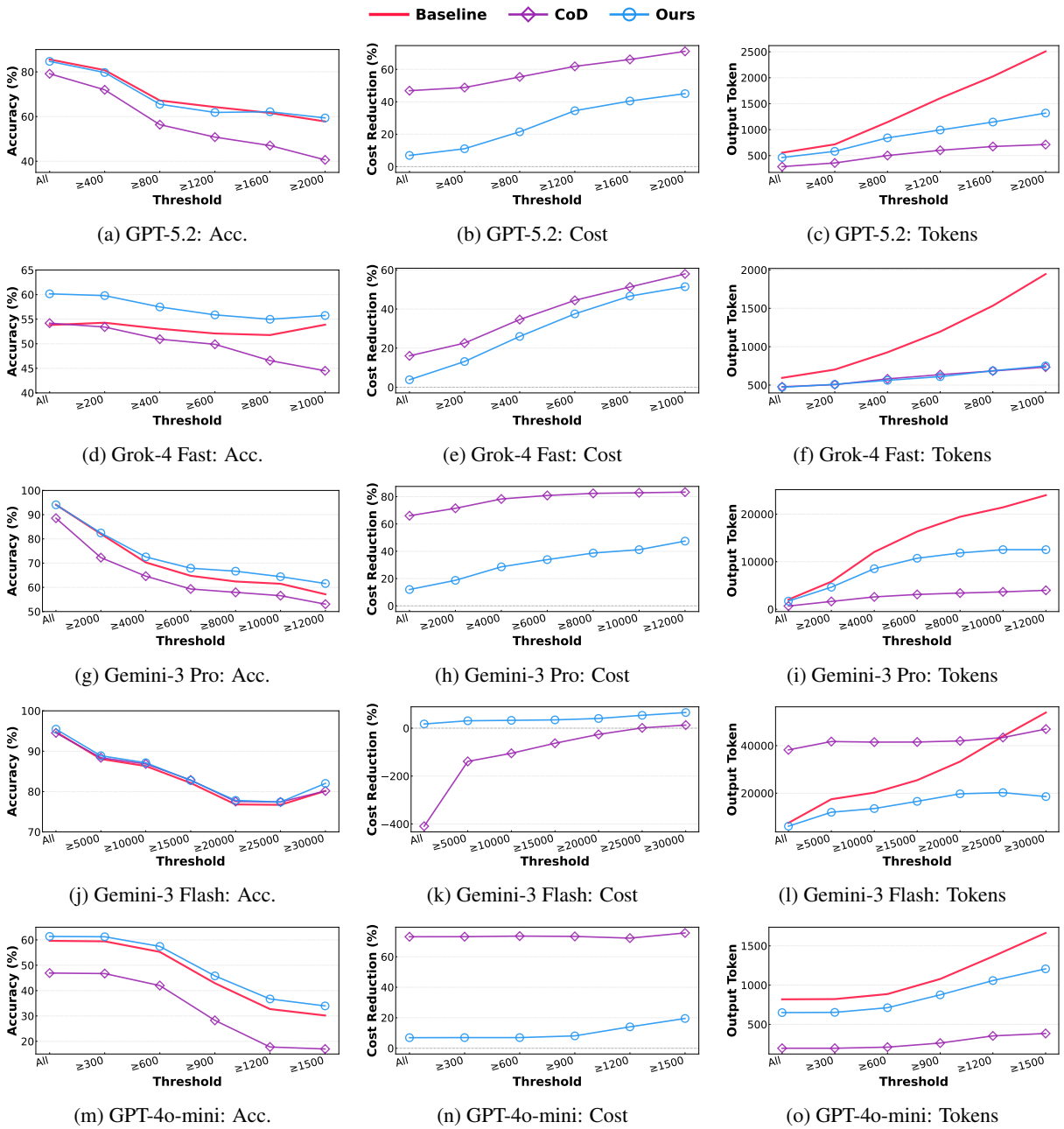
Figure 15: Full prompt template for No-Wait.

H Extended Comparison with Chain-of-Draft (CoD)

In this section, we extend the comparison between **Thinking with Reasoning Skills (TRS)** and the **Chain-of-Draft (CoD)** baseline to five additional models: **GPT-5.2**, **Grok-4-Fast**, **Gemini-3-Pro**, **Gemini-3-Flash**, and **GPT-4o-mini**. This analysis (visualized in Figure 16) complements Table 1 by breaking down performance across varying difficulty thresholds (θ), where a higher θ indicates problems that require longer reasoning traces from the baseline model.

The results highlight two critical advantages of TRS over CoD:

1. **Robustness on Hard Problems:** As difficulty increases (moving right on the x-axis), CoD frequently exhibits a sharp decline in accuracy (e.g., see GPT-4o-mini and Gemini-3-Pro panels). This confirms that simply forcing brevity often compromises reasoning depth on complex tasks. In contrast, TRS consistently maintains or improves upon baseline accuracy, effectively breaking the efficiency-accuracy trade-off.
2. **Consistent Cost Reduction:** While CoD achieves aggressive token reduction, it often does so at the expense of correctness. TRS achieves competitive cost reductions (e.g., significant savings on Gemini-3-Flash and GPT-5.2) while ensuring that the "thinking" process remains guided and correct via retrieved skills.



(a) GPT-5.2: Acc. (b) GPT-5.2: Cost (c) GPT-5.2: Tokens

(d) Grok-4 Fast: Acc. (e) Grok-4 Fast: Cost (f) Grok-4 Fast: Tokens

(g) Gemini-3 Pro: Acc. (h) Gemini-3 Pro: Cost (i) Gemini-3 Pro: Tokens

(j) Gemini-3 Flash: Acc. (k) Gemini-3 Flash: Cost (l) Gemini-3 Flash: Tokens

(m) GPT-4o-mini: Acc. (n) GPT-4o-mini: Cost (o) GPT-4o-mini: Tokens

Figure 16: Main results compared with CoD at different thresholds (DeepMath-103K).

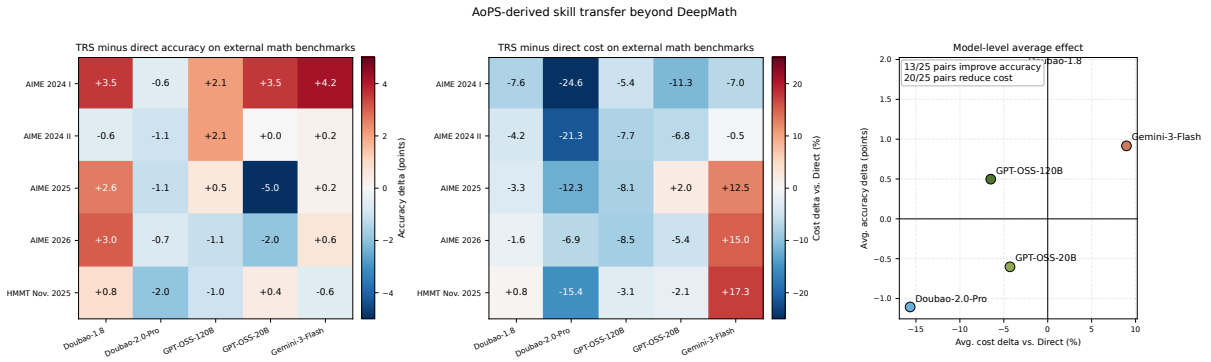


Figure 17: External contest-math transfer with the AoPS-derived skill bank. Left: TRS-minus-direct accuracy deltas. Middle: TRS-minus-direct cost deltas as percentages relative to Direct. Right: model-level average accuracy/cost-percentage trade-off. The overall pattern is mixed: 13 of 25 pairs improve accuracy and 20 of 25 reduce cost.

Benchmark	Doubao Direct → TRS	Doubao Cost Δ	OSS Direct → TRS	OSS Cost Δ
SST-2	$93.5 \rightarrow 92.5$	$+93.3\%$	$92.5 \rightarrow 92.5$	$+72.9\%$
CoLA	$86.5 \rightarrow 85.5$	$+102.8\%$	$82.0 \rightarrow 84.5$	$+52.4\%$
MRPC	$68.5 \rightarrow 67.0$	$+43.6\%$	$77.0 \rightarrow 79.0$	$+36.8\%$
AG News	$88.0 \rightarrow 87.5$	$+64.1\%$	$82.5 \rightarrow 85.0$	$+74.7\%$

Table 5: Non-reasoning stress test with a deliberately mismatched DeepMath skill library. Cost deltas are percentages relative to Direct. Skill injection does not cause catastrophic degradation, but it consistently increases cost and is not uniformly beneficial on short-form classification tasks.

Benchmark	Acc. Δ	Out Tok. Δ	Cost Δ
AIME 2024 I	+2.54	↓11.4%	↓10.8%
AIME 2024 II	+0.12	↓7.4%	↓6.3%
AIME 2025	-0.56	↑1.2%	↑4.9%
AIME 2026	-0.04	↑1.9%	↑7.3%
HMMT Nov. 2025	-0.48	↑3.0%	↑7.1%

Table 6: Benchmark-level averages on the external contest-math suite. Accuracy is reported in percentage points; output tokens and cost are relative changes against Direct. AIME 2024 I is the clearest positive transfer regime, while newer or harder sets show smaller or mixed effects.

I External Contest-Math Benchmark Transfer with AoPS Skills

The external benchmark study uses a *pure* AoPS-derived skill library consisting of 7,616 cards distilled from a broad contest-math corpus, including AHSME, AMC, AIME, USAMO, USAJMO, and IMO-style problems. The evaluation suite contains 120 questions in total: 15 from AIME 2024 I, 15 from AIME 2024 II, 30 from AIME 2025, 30 from AIME 2026, and 30 from HMMT November 2025. In this benchmark matrix, TRS builds a BM25 index over each skill card's question, topic, heuristic, and keywords, retrieves the top-1 match, and injects only the retrieved heuristic into the prompt. Figure 17 visualizes the full set of 25 model–benchmark results, including accuracy deltas, cost-percentage deltas, and the average trade-off at the model level, while Table 6 reports the benchmark-level averages. Importantly, all results reported in this section use the 7,616-card AoPS library rather than the later 7,736-card AoPS+benchmark merged library. Therefore, the transfer setting does not retrieve hints distilled from the benchmark suite itself.

The main takeaway is that AoPS-derived reasoning skills transfer *non-uniformly* across models and benchmarks. The strongest positive cells are Gemini-3-Flash on AIME 2024 I ($+4.16$), GPT-OSS-20B on AIME 2024 I ($+3.55$), and Doubao-1.8 on AIME 2024 I ($+3.54$), while the strongest negative cell is GPT-OSS-20B on AIME 2025 (-5.00). Because the direct and TRS settings use different repeat counts (32 versus 8), we interpret these results as descriptive evidence of transfer rather than as a formal hypothesis test. Overall, this analysis strengthens the camera-ready paper by showing that TRS is not confined to a single in-domain benchmark, although its benefits remain dependent on both target model and workload.

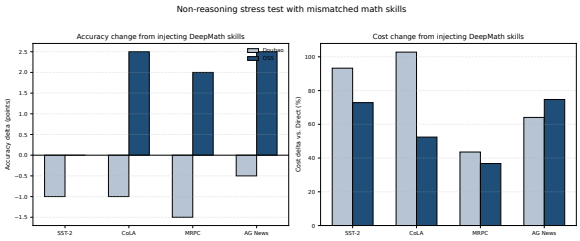


Figure 18: Accuracy and cost-percentage deltas in the non-reasoning stress test.

J Non-reasoning Stress Test

Table 5 and Figure 18 summarize a deliberately mismatched stress test in which DeepMath skills are injected into short-form classification tasks. This setting is intended as a robustness control rather than a recommended deployment scenario.

Across the four non-reasoning benchmarks, the average accuracy shift is -1.0 point for Doubao and $+1.75$ points for OSS, while the average cost increases by 75.9% and 59.2%, respectively. We therefore view this result primarily as a *robustness control*: it suggests that injecting mismatched reasoning skills does not lead to catastrophic degradation in accuracy, but it also provides no evidence that TRS should be enabled by default for mixed or low-reasoning workloads, especially given the substantial cost overhead.