

Share More, Search Less: Collaborative Parallel Thinking for Efficient Test-Time Scaling

Xinglin Wang^{1*}, Hao Lin^{1*}, Shaoxiong Feng^{2†}, Peiwen Yuan¹, Yiwei Li¹, Jiayi Shi¹, Yueqi Zhang¹, Chuyi Tan¹, Ji Zhang¹, Boyuan Pan², Yao Hu², Kan Li^{1†}

¹ School of Computer Science, Beijing Institute of Technology

² Xiaohongshu Inc

{wangxinglin, haolin20, peiwenyuan, liyiwei}@bit.edu.cn

{shijiayi, zhangyq, tanchuyi, zhangji, likan}@bit.edu.cn

shaoxiong.feng2023@gmail.com {panboyuan, xiahou}@xiaohongshu.com

Abstract

Test-Time Scaling (TTS) enhances the reasoning capabilities of large language models by allocating additional inference compute to explore the solution space. However, existing parallel TTS methods typically keep branches isolated during search: intermediate discoveries remain branch-private and cannot guide other branches in time. This information isolation causes substantial redundant exploration, as branches repeatedly rediscover information already found elsewhere and require more search steps to collect complete decision information needed to reach correct answers. To bridge this gap, we propose **Collaborative Parallel Thinking (CPT)**, a training-free inference framework that enables search-time information sharing across parallel branches. CPT extracts compact intermediate information from ongoing branches, maintains a deduplicated query-level information pool, and broadcasts pool entries through the input context, allowing each branch in subsequent search steps to reuse discoveries made by other branches rather than rediscover the same information. Empirically, experiments on HMMT and AIME show that CPT consistently pushes the accuracy-latency Pareto frontier beyond strong baselines across rollout budgets and model scales, demonstrating the effectiveness of search-time collaboration for efficient parallel TTS¹.

1 Introduction

Test-time scaling (TTS) has emerged as a powerful paradigm for improving the reasoning capabilities of large language models (Brown et al., 2024; Snell et al., 2024; Wu et al., 2025; Zhang et al., 2025; Liu et al., 2025a). By allocating additional computation at inference time, TTS enables extended exploration of the solution space, allowing models to

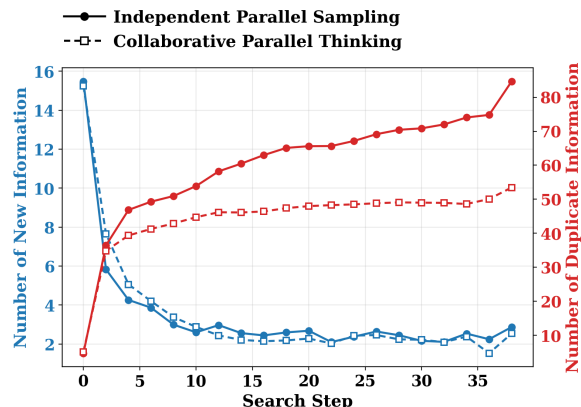


Figure 1: Information statistics during parallel search on HMMT24–25 using QWEN3-30B-A3B-THINKING-2507 with 64 parallel samples. At each 1024-token search step, we count newly discovered and duplicate information units, with details provided in Appendix B.1. As search proceeds, independent parallel sampling yields more duplicate information units per step, whereas Collaborative Parallel Thinking mitigates this redundancy while maintaining a comparable number of newly discovered information units.

gather more decision-relevant information, reduce uncertainty about the final answer, and thereby improve performance on complex tasks such as mathematical reasoning (OpenAI, 2024; Kimi Team et al., 2025; DeepSeek-AI et al., 2025). Rather than concentrating exploration in a single trajectory, parallel TTS generates multiple reasoning branches for the same problem simultaneously, accelerating solution-space exploration (Wang et al., 2023; Lightman et al., 2024; Fu et al., 2025; Zheng et al., 2026a,b).

While yielding promising performance gains, existing parallel TTS methods share a common bottleneck in information utilization (Wang et al., 2025e): branch-level discoveries remain confined to the trajectories that produce them, rather than being converted into shared decision information during search (Figure 2). Consequently, different

*Equal contribution.

†Corresponding author.

¹Our code and data have been released on <https://github.com/WangXinglin/CPT>.

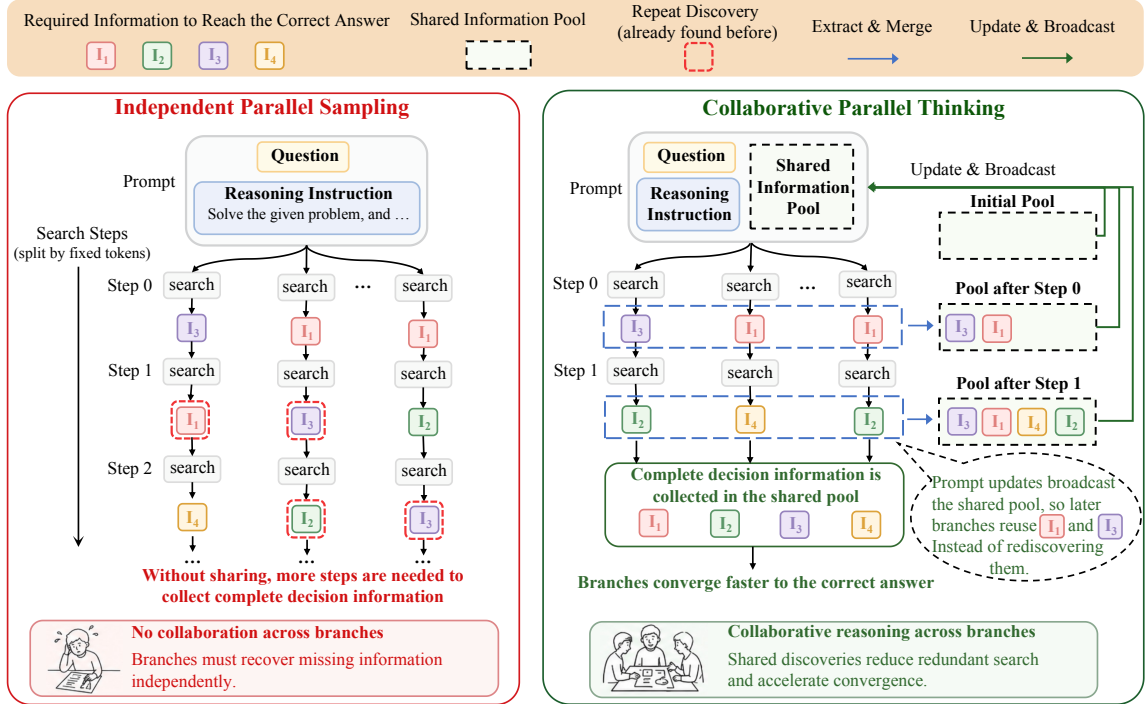


Figure 2: **Search-time collaboration reduces redundant parallel exploration.** (Left) Independent parallel sampling keeps branches isolated during search, so each branch must recover missing decision information independently and may repeatedly rediscover information found elsewhere. (Right) Collaborative Parallel Thinking (CPT) maintains a shared information pool that is provided to all branches. At fixed-token search steps, branch-level information is extracted, merged, and deduplicated into this pool. When broadcasting is active, pool entries are provided through the input context, allowing branches to reuse shared discoveries and converge faster once complete decision information is collected.

Metrics	Injection Ratio of Information (%)					
	0	20	40	60	80	100
Pass@1 (%)	48.95	53.34	54.82	55.89	56.20	55.86
Tokens	26714	16805	14128	12792	11924	11412
Latency (s)	395	269	235	223	219	220

Table 1: Effect of sharing branch-distributed information via offline injection on HMMT24–25 using QWEN3-4B-THINKING-2507 with 64 parallel samples. Each injection ratio corresponds to randomly sampling that proportion of entries from a deduplicated information pool extracted from parallel branches and injecting them into the initial context (details in Appendix B.2).

branches may repeatedly rediscover intermediate information that has already been found elsewhere, causing redundant exploration and slowing the accumulation of global decision information.

To validate this information-isolation bottleneck, we conduct two preliminary studies that examine both its induced redundancy and the utility of information that remains unshared across branches during search. To quantify the redundancy induced by information isolation, we track information statis-

tics at each fixed-token search step by counting newly discovered information units and duplicate information units. Figure 1 shows that, as search progresses, independent parallel sampling repeats information already discovered by other branches, wasting substantial computation for only diminishing marginal information gain. We further ask whether branch-distributed discoveries are useful once shared beyond the branches that originally found them. As shown in Table 1, offline injection of information sampled from a deduplicated branch-information pool supports more accurate reasoning and faster search convergence. Together, these results indicate that the bottleneck is not a lack of useful intermediate information within individual branches, but *the absence of a search-time mechanism for pooling and broadcasting branch-private discoveries as reusable global decision information*.

Motivated by this bottleneck, we propose **Collaborative Parallel Thinking (CPT)**, a training-free inference method that enables parallel reasoning branches to share and reuse decision-relevant

information during search. Rather than treating branch-level discoveries as private byproducts of independent branches, CPT extracts compact information from ongoing branches and maintains it in a deduplicated query-level pool. At fixed-token search steps, pool entries are broadcast through the input context, so subsequent generation chunks can reuse discoveries made by other branches while preserving each branch’s private reasoning history. To avoid premature convergence of exploration directions, CPT further adopts an adaptive broadcast schedule that controls when to share: branches first explore independently, broadcasting starts after marginal new-information gain drops substantially, and synchronization stops once further sharing offers limited benefit.

We evaluate CPT with three policy models across a broad range of rollout budgets on challenging mathematical reasoning benchmarks, including HMMT24, HMMT25 (Balunović et al., 2025), AIME25, and AIME26 (Dekoninck et al., 2026). CPT consistently pushes the accuracy–latency Pareto frontier beyond strong baselines, achieving higher accuracy at matched wall-clock latency. Additional experiments on GPQA-Diamond (Rein et al., 2024) and LiveCodeBench-v6 (Jain et al.) further demonstrate that the benefits of search-time collaboration generalize beyond contest mathematics to scientific reasoning and code generation.

To summarize, our main contributions are:

- We identify and validate an information-isolation bottleneck in parallel TTS: branch-private discoveries remain unshared during search, causing redundant rediscovery and slowing convergence toward correct answers.
- We propose Collaborative Parallel Thinking (CPT), a training-free inference method that enables parallel branches to share and reuse decision-relevant information during search through compact extraction, deduplicated pooling, shared-context broadcasting, and adaptive broadcast scheduling.
- We empirically validate CPT with three policy models across diverse rollout budgets, demonstrating a stronger accuracy–latency Pareto frontier than strong baselines on HMMT24–25 and AIME25–26, with additional results on GPQA-Diamond and LiveCodeBench-v6 supporting generalization beyond contest mathematics.

2 Related Work

Test-Time Scaling. The paradigm of scaling test-time compute has emerged as a critical avenue for enhancing reasoning capabilities (OpenAI, 2024; DeepSeek-AI et al., 2025; Wu et al., 2025). Existing methods differ in how this compute is organized (Zhang et al., 2025). Sequential refinement methods extend or revise a single reasoning trajectory over time (Wei et al., 2022; Madaan et al., 2023; Shinn et al., 2023), while structured search methods such as Tree-of-Thoughts, MCTS-style decoding, and verifier-guided search introduce explicit branching, scoring, and pruning over candidate prefixes (Yao et al., 2023; Liu et al., 2023; Wan et al., 2024; Lightman et al., 2024; Wang et al., 2025c). These methods improve the quality of test-time search by steering computation toward more promising reasoning states, but their reliance on sequential refinement, intermediate evaluation, or verifier-guided control can lengthen the critical inference path and introduce additional latency overhead (Snell et al., 2024; Wang et al., 2025d; Zheng et al., 2026a,b; Wang et al., 2026). Parallel TTS instead generates multiple reasoning branches for the same problem simultaneously, accelerating solution-space exploration through width scaling (Wang et al., 2023; Li et al., 2024a; Wang et al., 2025b; Fu et al., 2025; Zheng et al., 2025; Dai et al., 2026). However, standard parallel sampling typically keeps branches independent during search and aggregates their outputs only after generation, leaving intermediate discoveries isolated within individual branches. The closest attempt to move beyond this independent formulation is LeaP (Luo et al., 2025), which introduces peer-path routing for self-correction and helps reasoning paths recover from poor prefixes during inference. Unlike LeaP’s path-specific routing for self-correction, CPT is efficiency-oriented: it maintains a deduplicated query-level shared information pool and broadcasts it globally through the input context, turning branch-private discoveries into reusable decision information to reduce redundant rediscovery.

Collaborative Reasoning with LLMs. Collaboration among LLM instances has been explored through multi-agent debate and discussion, where agents exchange arguments, critiques, or proposals to encourage divergent reasoning, consensus formation, or improved final decisions (Liang et al., 2024; Chen et al., 2024; Estornell and Liu, 2024; Liu et al., 2025b). Recent analyses further exam-

ine when such interaction is beneficial compared with single-agent prompting or voting-based aggregation (Wang et al., 2024; Choi et al., 2026). To improve efficiency, follow-up work studies sparse communication topology, token-efficient debate, group discussion, and selective debate triggering (Li et al., 2024b; Zeng et al., 2025; Liu et al., 2024; Eo et al., 2025). Beyond debate, Mixture-of-Agents and scalable multi-agent collaboration frameworks aggregate or orchestrate multiple model instances to exploit complementary strengths (Wang et al., 2025a; Qian et al., 2025). These works organize collaboration mainly at the agent, model, or response level. In contrast, CPT studies collaboration among parallel reasoning branches of the same query during test-time search, sharing compact intermediate information through a deduplicated query-level pool to reduce redundant rediscovery.

3 Methodology

Collaborative Parallel Thinking (CPT) is motivated by a simple principle: parallel branches should share useful discoveries during search instead of repeatedly rediscovering information found elsewhere. We therefore turn parallel TTS from isolated branch-wise exploration into a search-time information-sharing process, where compact discoveries are extracted into a deduplicated query-level pool and broadcast through the input context to guide subsequent exploration. To keep sharing lightweight and stable, CPT synchronizes branches only at fixed-token search steps while preserving their private reasoning histories.

The overall workflow of CPT comprises three coordinated components: **Collaborative Parallel Search**, which synchronizes branches at fixed-token steps while preserving their private reasoning histories; **Shared Information Pooling**, which extracts compact information, maintains a deduplicated query-level pool, and broadcasts pool entries through the input context; and **Adaptive Broadcast Scheduling**, which decides when to start and stop broadcasting according to the relative rate of newly discovered information. A detailed algorithmic description is provided in Algorithm 1 in Appendix A.

3.1 Collaborative Parallel Search

Given a problem x and a policy model π , CPT launches K reasoning branches in parallel. At

search step t , each branch i maintains its own private reasoning history h_i^t , while the system maintains a query-level information pool $\mathcal{P}_t$. Each search step corresponds to a fixed-token generation chunk of length C . During a step, every unfinished branch continues generation from its own private history, optionally conditioned on the current shared information block if broadcasting is active. After the chunk is generated, CPT temporarily synchronizes the branches and passes their newly generated segments to the information-pooling stage.

A key design choice is that CPT only synchronizes at step boundaries. It does not rewrite the generated reasoning trace, nor does it inject shared content into the middle of a branch’s private chain. Thus, each branch preserves its own exploration trajectory, while the shared information block serves as an additional context for subsequent search steps. This design preserves branch-level exploration while enabling search-time reuse of information discovered by other branches.

3.2 Shared Information Pooling

Raw reasoning trajectories are too long and noisy to be shared directly. CPT therefore asks the same policy model π to extract compact search information from each branch, such as intermediate conclusions, constraints, observations, counterexamples, and useful checks. Given the problem x , the current branch history h_i^t , and the newly generated segment Δh_i^t , the extractor outputs a small set of textual information units:

$$\mathcal{Z}_i^t = \text{Extract}_\pi(x, h_i^t, \Delta h_i^t).$$

This extraction step converts unstructured reasoning traces into compact information units that can be pooled and broadcast under a limited context budget.

All extracted information units are maintained in a deduplicated query-level pool $\mathcal{P}_t$. When updating the pool, CPT filters out candidates that are semantically similar to existing entries under an embedding-similarity threshold τ_{dup} , where similarities are computed with text embeddings $\phi(\cdot)$. This keeps the pool compact and information-dense, preventing repeated branch discoveries from consuming the shared context budget.

When broadcasting is active, CPT samples at most M entries from the deduplicated pool and serializes them into the shared-information section of the input context. Before the next search step, the

same updated context is provided to all branches, while each branch keeps its own private reasoning history. In this way, branch-private discoveries become reusable global decision information that can guide subsequent parallel exploration.

3.3 Adaptive Broadcast Scheduling

Broadcasting is helpful only when the shared pool contains information that can influence later search. Starting too early may reduce exploration diversity before enough local discoveries have been collected, while continuing too late brings little new information but still incurs synchronization and prefilling overhead. CPT therefore uses the rate of newly admitted pool entries as a lightweight proxy for marginal information gain.

Let n_t be the number of new entries added to the pool at search step t . To make this signal stable, CPT averages it over fixed windows of W search steps. For the j -th window $\mathcal{W}_j$, we compute

$$g_j = \frac{1}{|\mathcal{W}_j|} \sum_{t \in \mathcal{W}_j} n_t, \quad r_j = \frac{g_j}{g_1 + \epsilon},$$

where g_1 is the average new-information count in the first window and ϵ is a small constant for numerical stability. The ratio r_j measures how much the current marginal information gain has declined relative to the initial exploration stage. Using the first window as a problem-specific reference allows CPT to adapt to different reasoning lengths and information densities.

CPT uses two thresholds, a start threshold τ_{start} and a stop threshold τ_{stop} , with $\tau_{\text{stop}} < \tau_{\text{start}}$, to control the broadcast schedule. In the **probe phase**, branches explore independently while still writing extracted information into the pool. When $r_j < \tau_{\text{start}}$, CPT enters the **broadcast phase** and begins providing the shared pool to all branches at subsequent search steps. When $r_j < \tau_{\text{stop}}$, CPT enters the **free-run phase**, stops further synchronization, and lets each branch decode until termination. This schedule activates sharing when independent exploration starts to yield fewer new discoveries, and stops sharing when the marginal benefit of additional broadcasts becomes low.

4 Experiments

4.1 Experimental Setup

Benchmarks and Models. We evaluate the proposed CPT strategy on challenging mathematical reasoning benchmarks, including HMMT24,

HMMT25 (Balunović et al., 2025), AIME25, and AIME26 (Dekoning et al., 2026). These benchmarks require complex multi-step reasoning and serve as reliable testbeds for evaluating whether test-time methods can efficiently explore solution spaces under practical latency budgets. We primarily evaluate CPT on models specialized for complex reasoning tasks, including QWEN3-4B-THINKING-2507, QWEN3-30B-A3B-THINKING-2507 (Yang et al., 2025) and GPT-OSS-20B (Agarwal et al., 2025). To assess generalization beyond mathematical reasoning, we further evaluate GPQA-Diamond (Rein et al., 2024) for scientific question answering and LiveCodeBench-v6 (Jain et al.) for code generation (See Appendix F).

Baselines. We compare CPT against representative parallel test-time scaling and efficiency strategies: (1) **Base Parallel Sampling**, which directly samples multiple reasoning branches from the policy model without any additional search-time mechanism, reflecting the model’s raw parallel-sampling capability; (2) **DeepConf** (Fu et al., 2025), which uses confidence signals to identify and retain high-quality reasoning traces; and (3) **LeaP** (Luo et al., 2025), the closest cross-path interaction baseline, which uses peer-path routing for self-correction during inference. Since our focus is latency-efficient parallel TTS, we do not compare against sequential or structured search methods such as Self-Refine, Tree-of-Thoughts, and MCTS-style search, whose iterative refinement or lookahead procedures often require additional serial computation and introduce substantial end-to-end latency overhead.

Evaluation Protocol. Unless otherwise specified, we evaluate each method under a range of parallel rollout budgets $K \in \{8, 16, 32, 64, 128\}$, where larger K corresponds to scaling test-time computation by launching more reasoning branches. For each budget, we measure wall-clock latency from the start of parallel decoding until all reasoning branches finish decoding, and report two accuracy metrics after answer normalization. **Pass@1** measures the average correctness of individual sampled branches, while **MV@K** measures the majority-vote accuracy obtained by aggregating the final answers produced under the corresponding K -branch budget. To ensure statistical reliability, we conduct 8 independent runs for all experiments, and report averaged results. Since our focus is latency-efficient parallel TTS, we use wall-clock latency as

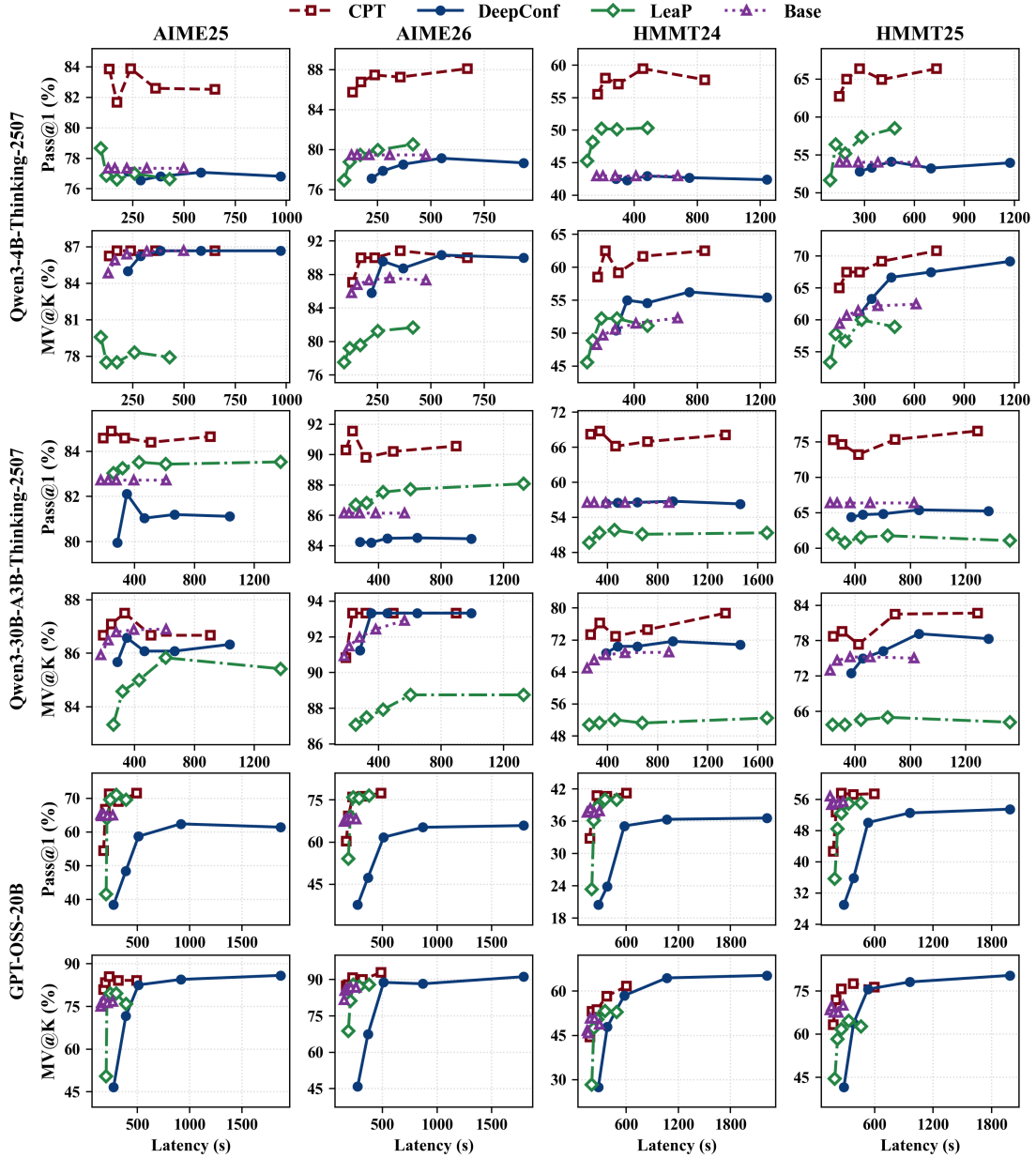


Figure 3: Accuracy–Latency comparison across models and benchmarks under different rollout budgets.

the primary compute budget and compare methods by their accuracy–latency Pareto frontier. For CPT, the reported latency is end-to-end and includes all operations executed by the method, including parallel generation, search-information extraction, semantic deduplication, input-context broadcast, and final answer aggregation. Prompt templates are provided in Appendix I.

Implementation Details. All experiments are executed in parallel on a cluster with 512 NVIDIA H800 GPUs, where each individual run is allocated to 8 GPUs. For CPT, the same policy model is used for both reasoning generation and search-information extraction, without external reward

models, verifiers, or task-specific training. Unless otherwise specified, CPT uses a search-step length of 2048 tokens and samples at most $M = 512$ pool entries for each broadcast. For semantic de-duplication, we encode extracted information units with all-MiniLM-L6-v2² and use a cosine-similarity threshold of $\tau_{\text{dup}} = 0.75$. For adaptive broadcast scheduling, we compute the relative new-information gain over windows of $W = 3$ search steps, with $\tau_{\text{start}} = 0.4$ and $\tau_{\text{stop}} = 0.1$. For Qwen3-Thinking Models, we follow the official recommended sampling hyperparameters: temper-

²<https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>

ature $T = 0.6$, $\text{top-}p = 0.95$, $\text{top-}k = 20$, with a maximum generation length of 38k tokens. All CPT hyperparameters are kept fixed across benchmarks, models, and rollout budgets unless explicitly ablated.

4.2 Main Results

The main results across mathematical reasoning benchmarks and model scales are summarized in Figure 3. CPT consistently establishes a stronger accuracy–latency Pareto frontier than the compared baselines across different rollout budgets, achieving higher accuracy under comparable wall-clock latency. This advantage is observed on all policy models, and across AIME and HMMT benchmarks, indicating that search-time information sharing remains effective across model scales and problem distributions. The simultaneous gains in Pass@1 and MV@K indicate that CPT improves individual branch quality and the overall effectiveness of parallel exploration through information sharing, rather than simply increasing final-answer consistency by prematurely aligning branch trajectories. Overall, our results suggest that reducing redundant rediscovery through search-time information sharing is an effective mechanism for latency-efficient parallel TTS.

4.3 Analysis

Information Redundancy Reduction. Beyond the accuracy–latency frontier, we analyze how CPT changes the information content of parallel search. As shown in Figure 1, independent parallel sampling yields more duplicate information units per search step as search proceeds, while the number of newly discovered information units decreases. This indicates that later exploration increasingly spends computation rediscovering information already found by other branches. In contrast, CPT reduces per-step duplicate discovery while maintaining a comparable level of new-information discovery. This suggests that the gains of CPT do not merely come from forcing branches to agree earlier, but from reducing the residual search burden through cross-branch information sharing.

Effect of Adaptive Broadcast Scheduling. Table 2 analyzes the effect of the adaptive start and stop thresholds. When fixing $\tau_{\text{start}} = 0.4$, setting $\tau_{\text{stop}} = 0$ disables early stopping of synchronization, while larger τ_{stop} values stop broadcasting earlier. The results show that moderate early stop-

ping generally provides a better accuracy–latency trade-off than either never stopping or stopping too aggressively, suggesting that continued synchronization becomes less useful once marginal new-information gain is low. When fixing $\tau_{\text{stop}} = 0.1$, setting $\tau_{\text{start}} = 1.0$ starts broadcasting immediately, while smaller values allow branches to explore independently before sharing begins. Immediate broadcasting does not yield the best accuracy, indicating that overly early sharing can prematurely align exploration directions. Overall, these results support our adaptive scheduling design: CPT should first preserve independent exploration, then activate sharing when redundant discovery increases, and finally stop synchronization once additional broadcasts provide limited benefit.

Hyperparameter Sensitivity. We analyze the sensitivity of CPT to two key hyperparameters: the broadcast size M and the semantic de-duplication threshold τ_{dup} . As shown in Table 4, increasing M generally improves Pass@1 and reduces both generated tokens and latency, indicating that providing more shared information can help branches avoid redundant exploration and converge more efficiently. However, the gain saturates when M becomes large, and $M = 512$ achieves the best accuracy in our setting. Table 5 shows that CPT is relatively stable across a range of de-duplication thresholds, with $\tau_{\text{dup}} = 0.75$ giving the best Pass@1. A threshold that is too low may filter out useful but related information, while a threshold that is too high may admit redundant entries into the shared pool. We therefore use $M = 512$ and $\tau_{\text{dup}} = 0.75$ as the default configuration in our main experiments.

Component-wise Runtime Cost. Table 3 reports the component-wise latency and FLOPs of CPT at $N = 64$. Parallel sampling remains the dominant source of wall-clock latency, while information extraction and semantic de-duplication/filtering add moderate overhead. The FLOPs of information extraction are small compared with sampling, suggesting that the extraction itself is not the main computational bottleneck. Instead, the main FLOPs trade-off arises from prompt-level broadcasting, which may require re-prefilling updated contexts in standard decoding implementations. This motivates future cache-aware or attention-level sharing mechanisms to further reduce the computational overhead of CPT.

Benchmark	Metrics	Stop Threshold τ_{stop}				Start Threshold τ_{start}				
		0	0.10	0.20	0.30	0.30	0.40	0.50	0.60	1.00
HMMT24	Pass@1 (%)	57.86	58.35	56.60	56.75	53.94	58.35	58.78	56.74	53.05
	Tokens ($\times 10^6$)	1.52	1.57	1.51	1.54	1.62	1.57	1.48	1.45	1.43
	Latency (s)	482	457	413	395	436	457	453	447	449
HMMT25	Pass@1 (%)	66.94	66.79	65.33	64.58	59.67	66.79	65.28	66.22	63.37
	Tokens ($\times 10^6$)	1.43	1.47	1.42	1.42	1.50	1.47	1.38	1.34	1.29
	Latency (s)	410	403	380	354	380	403	400	395	393

Table 2: Sensitivity analysis of adaptive broadcast scheduling thresholds using QWEN3-4B-THINKING-2507 with 64 parallel samples. For the stop-threshold sweep, $\tau_{\text{start}} = 0.4$ is fixed; for the start-threshold sweep, $\tau_{\text{stop}} = 0.1$ is fixed.

Benchmark	Model	Latency			FLOPs	
		Sampling	Info Extract	Dedup. & Filter	Sampling	Info Extract
AIME25	QWEN3-4B	307.90	35.65	18.16	112.43	1.66
	QWEN3-30B-A3B	420.37	61.59	29.44	81.99	1.70
AIME26	QWEN3-4B	291.00	39.79	21.92	107.11	1.50
	QWEN3-30B-A3B	415.13	50.42	24.86	82.02	1.80
HMMT24	QWEN3-4B	368.43	56.75	26.95	138.52	1.62
	QWEN3-30B-A3B	607.90	83.55	34.46	141.12	2.64
HMMT25	QWEN3-4B	331.72	44.42	22.39	120.02	1.50
	QWEN3-30B-A3B	582.40	75.03	32.27	132.76	2.35

Table 3: Component-wise latency and FLOPs of CPT with 64 parallel samples. Latency is reported in seconds and FLOPs are reported in PFLOPs per question.

Metrics	Broadcast Size M					
	32	64	128	256	512	1024
Pass@1 (%)	61.65	61.93	63.68	64.03	66.79	65.32
Tokens ($\times 10^6$)	1.53	1.48	1.43	1.42	1.47	1.42
Latency (s)	429	411	408	407	402	385

Table 4: Effect of broadcast size M on HMMT25 using QWEN3-4B-THINKING-2507 with 64 parallel samples. We vary the maximum number of pool entries inserted into the shared-information section at each broadcast.

Metrics	Similarity Threshold τ_{dup}				
	0.65	0.70	0.75	0.80	0.85
Pass@1 (%)	62.60	64.57	66.79	65.92	66.66
Tokens ($\times 10^6$)	1.51	1.48	1.47	1.46	1.47
Latency (s)	383	385	402	400	405

Table 5: Effect of similarity threshold τ_{dup} on HMMT25 using QWEN3-4B-THINKING-2507 with 64 parallel samples. We vary the semantic de-duplication threshold used to filter information units before adding them to the shared pool.

Generated Tokens and FLOPs. Although our main comparison uses wall-clock latency as the primary compute budget, we also examine generated tokens and FLOPs as auxiliary cost metrics in Appendix C. The accuracy–tokens results (Figure 4) show that CPT achieves a stronger token-efficiency frontier, indicating that search-time information sharing reduces unnecessary decoding spent on rediscovering information already found by other branches. The FLOPs comparison (Figure 5) reveals a different trade-off: as CPT implements sharing through input-context updates, standard decoding implementations may require re-prefilling the updated context, introducing additional computa-

tion. Thus, CPT improves latency-efficient parallel exploration and reduces redundant generated tokens, while more cache-aware or attention-level sharing mechanisms are needed to further improve FLOPs efficiency.

5 Conclusions

In this work, we identify and validate an information-isolation bottleneck in parallel test-time scaling through preliminary studies: branch-private discoveries remain unshared during search, leading to redundant rediscovery and slower convergence toward correct answers. To bridge this

gap, we introduce Collaborative Parallel Thinking (CPT), a training-free inference framework that enables search-time information sharing across parallel branches. CPT builds a shared information pool by extracting compact intermediate information from ongoing branches and merging overlapping entries. Pool entries are then broadcast through the input context, turning branch-private discoveries into reusable global decision information. Empirical evaluations on challenging mathematical reasoning benchmarks demonstrate that CPT establishes a stronger accuracy–latency Pareto frontier than strong baselines across rollout budgets and model scales. Overall, our findings highlight that the efficiency of parallel TTS depends not only on launching more reasoning branches, but also on enabling branches to share and reuse decision-relevant information during search.

Limitations

Collaborative Parallel Thinking (CPT) uses a fixed-token synchronous protocol to make branch discoveries reusable during parallel search, balancing timely information reuse against synchronization and context-update overhead. Since the shared information pool is updated only after each search step, discoveries made within a step cannot influence other branches until the next step, so duplicate discoveries may still occur before the next broadcast. In addition, broadcasting the updated pool through the input context is not free: in standard decoding implementations, it can require refilling the updated context, introducing additional FLOPs and latency overhead. Future work could reduce this cost by designing attention-level or cache-aware mechanisms that reuse prior computation, support lower-cost context updates, or enable finer-grained sharing without repeatedly recomputing the full prefix.

Ethics Statement

All of the datasets used in this study were publicly available, and no annotators were employed for our data collection. We confirm that the datasets we used did not contain any harmful content and were consistent with their intended use (research). We have cited the datasets and relevant works used in this study.

Acknowledgements

This research was supported by the Beijing Natural Science Foundation (No. 4262065, 4222037, and L181010).

References

2025. Nemo rl: A scalable and efficient post-training library. <https://github.com/NVIDIA-NeMo/RL>. GitHub repository.
- Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.
- Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. 2025. [Matharena: Evaluating llms on uncontaminated math competitions](#).
- Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.
- Justin Chen, Swarnadeep Saha, and Mohit Bansal. 2024. Reconcile: Round-table conference improves reasoning via consensus among diverse llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7066–7085.
- Hyeong Kyu Choi, Jerry Zhu, and Sharon Li. 2026. Debate or vote: Which yields better decisions in multi-agent large language models? *Advances in Neural Information Processing Systems*, 38:101732–101764.
- Runpeng Dai, Tong Zheng, Rui Liu, Chengsong Huang, and Hongtu Zhu. 2026. Small rl controller, large language model: RL-guided adaptive sampling for test-time scaling. *arXiv preprint arXiv:2606.03102*.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-rl: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Jasper Dekoninck, Nikola Jovanović, Tim Gehringer, Kári Rognvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. 2026. [Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms](#).
- Sugyeong Eo, Hyeonseok Moon, Evelyn Hayoon Zi, Chanjun Park, and Heuseok Lim. 2025. Debate

- only when necessary: Adaptive multiagent collaboration for efficient llm reasoning. *arXiv preprint arXiv:2504.05047*.
- Andrew Estornell and Yang Liu. 2024. Multi-llm debate: Framework, principals, and interventions. *Advances in Neural Information Processing Systems*, 37:28938–28964.
- Yichao Fu, Xuwei Wang, Yuandong Tian, and Jiawei Zhao. 2025. [Deep think with confidence](#). *CoRR*, abs/2508.15260.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fan-jia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Live-codebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*.
- Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. 2025. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.
- Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. 2024a. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In *The Twelfth International Conference on Learning Representations*.
- Yunxuan Li, Yibing Du, Jiageng Zhang, Le Hou, Peter Grabowski, Yeqing Li, and Eugene Ie. 2024b. Improving multi-agent debate with sparse communication topology. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7281–7294.
- Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2024. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 17889–17904.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations*.
- Jiacheng Liu, Andrew Cohen, Ramakanth Pasunuru, Yejin Choi, Hannaneh Hajishirzi, and Asli Celikyilmaz. 2023. Don’t throw away your value model! generating more preferable text with value-guided monte-carlo tree search decoding. *arXiv preprint arXiv:2309.15028*.
- Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. 2025a. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. *arXiv preprint arXiv:2502.06703*.
- Tongxuan Liu, Xingyu Wang, Weizhe Huang, Wenjiang Xu, Yuting Zeng, Lei Jiang, Hailong Yang, and Jing Li. 2024. Groupdebate: Enhancing the efficiency of multi-agent debate using group discussion. *arXiv preprint arXiv:2409.14051*.
- Yexiang Liu, Jie Cao, Zekun Li, Ran He, and Tieniu Tan. 2025b. Breaking mental set to improve reasoning through diverse multi-agent debate. In *The Thirteenth International Conference on Learning Representations*.
- Tongxu Luo, Wenyu Du, Jiayi Bi, Stephen Chung, Zhengyang Tang, Hao Yang, Min Zhang, and Benyou Wang. 2025. Learning from peers in reasoning models. *arXiv preprint arXiv:2505.07787*.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 46534–46594.
- OpenAI. 2024. [Learning to reason with llms](#).
- Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, et al. 2025. Scaling large language model-based multi-agent collaboration. In *International Conference on Learning Representations*, volume 2025, pages 41488–41505.
- 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*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.
- 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*.
- Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2024. AlphaZero-like tree-search can guide large language model decoding and training. In *International Conference on Machine Learning (ICML)*, volume 235, pages 49890–49920.
- Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y Zou. 2025a. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations*, volume 2025, pages 33944–33963.

- Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. 2024. Rethinking the bounds of llm reasoning: Are multi-agent discussions the key? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6106–6131.
- Xinglin Wang, Shaoxiong Feng, Yiwei Li, Peiwen Yuan, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. 2025b. Make every penny count: Difficulty-adaptive self-consistency for cost-efficient reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6919–6932.
- Xinglin Wang, Yiwei Li, Shaoxiong Feng, Peiwen Yuan, Yueqi Zhang, Jiayi Shi, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. 2025c. [Every rollout counts: Optimal resource allocation for efficient test-time scaling](#). *ArXiv*, abs/2506.15707.
- Xinglin Wang, Jiayi Shi, Shaoxiong Feng, Peiwen Yuan, Yiwei Li, Yueqi Zhang, Chuyi Tan, Ji Zhang, Boyuan Pan, Yao Hu, et al. 2026. Do not waste your rollouts: Recycling search experience for efficient test-time scaling. *arXiv preprint arXiv:2601.21684*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. [Self-consistency improves chain of thought reasoning in language models](#). In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.
- Zili Wang, Tianyu Zhang, Haoli Bai, Lu Hou, Xianzhi Yu, Wulong Liu, Shiming Xiang, and Lei Zhu. 2025d. Faster and better llms via latency-aware test-time scaling. *arXiv preprint arXiv:2505.19634*.
- Ziqi Wang, Boye Niu, Zipeng Gao, Zhi Zheng, Tong Xu, Linghui Meng, Zhongli Li, Jing Liu, Yilong Chen, Chen Zhu, et al. 2025e. A survey on parallel reasoning. *arXiv preprint arXiv:2510.12164*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2025. Inference scaling laws: An empirical analysis of compute-optimal inference for llm problem-solving. In *The Thirteenth International Conference on Learning Representations*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 40 others. 2025. [Qwen3 technical report](#). *CoRR*, abs/2505.09388.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.
- Yuting Zeng, Weizhe Huang, Lei Jiang, Tongxuan Liu, Xitai Jin, Chen Tianying Tiana, Jing Li, and Xiaohua Xu. 2025. S2-mad: Breaking the token barrier to enhance multi-agent debate efficiency. 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 9393–9408.
- Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenye Hua, Haolun Wu, Zhihan Guo, Yufei Wang, Niklas Muennighoff, et al. 2025. A survey on test-time scaling in large language models: What, how, where, and how well? *arXiv preprint arXiv:2503.24235*.
- Tong Zheng, Chengsong Huang, Runpeng Dai, Yun He, Rui Liu, Xin Ni, Huiwen Bao, Kaishen Wang, Hongtu Zhu, Jiaxin Huang, et al. 2026a. Parallel-probe: Towards efficient parallel thinking via 2d probing. *arXiv preprint arXiv:2602.03845*.
- Tong Zheng, Haolin Liu, Chengsong Huang, Huiwen Bao, Sheng Zhang, Rui Liu, Runpeng Dai, Ruibo Chen, Chenxi Liu, Tianyi Xiong, et al. 2026b. Llms improving llms: Agentic discovery for test-time scaling. *arXiv preprint arXiv:2605.08083*.
- Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, and Dong Yu. 2025. Parallel-r1: Towards parallel thinking via reinforcement learning. *arXiv preprint arXiv:2509.07980*.

A Collaborative Parallel Thinking Algorithm

Algorithm 1 provides the full procedure of Collaborative Parallel Thinking (CPT). For a problem x , CPT maintains K private branch histories $\{h_i\}_{i=1}^K$ and a query-level shared information pool $\mathcal{P}$. The procedure alternates between fixed-token parallel generation and pool update: at each search step, unfinished branches decode up to C tokens conditioned on the current broadcast set $\mathcal{B}$, after which the same policy model extracts compact information from the newly generated segments. Candidate information units are added to $\mathcal{P}$ only when they are not semantically similar to existing entries, keeping the pool as a compact and deduplicated state of branch discoveries.

The broadcast schedule is controlled by the new-information record $\mathcal{N}$. CPT starts in the PROBE mode, where branches explore independently while still writing extracted information into the pool. Once the windowed relative information gain falls below τ_{start} , CPT enters the BROADCAST mode and samples pool entries to all branches through the shared-information section of the input context. When the relative gain further falls below τ_{stop} , CPT enters the FREERUN mode and lets the remaining branches decode until EOS or L_{max} without further synchronization. This implements bounded search-time sharing while preserving each branch’s private reasoning history.

Algorithm 1 Collaborative Parallel Thinking (CPT)

Require: Problem x , model π , branch number K , search-step length C , maximum length $L_{\max}$, broadcast size M , deduplication threshold τ_{dup} , window size W , start threshold τ_{start} , stop threshold τ_{stop} , numerical constant ϵ .

Ensure: Final reasoning branches $\{h_i\}_{i=1}^K$.

```
1: Initialize branch histories  $h_i \leftarrow \emptyset$  for all  $i \in [K]$ ; shared pool  $\mathcal{P} \leftarrow \emptyset$ ; mode  $\leftarrow$  PROBE; new-information record  $\mathcal{N} \leftarrow []$ ;
   reference gain  $g_{\text{ref}} \leftarrow$  NONE.
2: while some branch is unfinished, i.e., it has reached neither EOS nor  $L_{\max}$  do
3:   // Component 1: Collaborative Parallel Search (Sec. 3.1)
4:   if mode = BROADCAST then
5:      $\mathcal{B} \leftarrow \mathcal{P}$  if  $|\mathcal{P}| \leq M$ ; otherwise sample  $M$  entries from  $\mathcal{P}$ .
6:   else
7:      $\mathcal{B} \leftarrow \emptyset$ .
8:   end if
9:   for each unfinished branch  $i \in [K]$  in parallel do
10:     $\Delta h_i \sim \pi(\cdot \mid \text{Prompt}(x, h_i, \mathcal{B}))$  for at most  $C$  tokens, or until EOS or  $L_{\max}$  is reached.
11:     $h_i \leftarrow h_i \oplus \Delta h_i$ .
12:   end for
13:   if all branches have reached EOS or  $L_{\max}$  then
14:     break
15:   end if
16:   // Component 2: Shared Information Pooling (Sec. 3.2)
17:    $n \leftarrow 0$ .
18:   for each branch  $i$  with newly generated segment  $\Delta h_i$  do
19:      $\mathcal{Z}_i \leftarrow \text{Extract}_{\pi}(x, h_i, \Delta h_i)$ .
20:     for each  $z \in \mathcal{Z}_i$  do
21:       if  $\mathcal{P} = \emptyset$  then
22:          $\mathcal{P} \leftarrow \mathcal{P} \cup \{z\}$ ;  $n \leftarrow n + 1$ .
23:       else
24:          $s^* \leftarrow \max_{p \in \mathcal{P}} \cos(\phi(z), \phi(p))$ .
25:         if  $s^* < \tau_{\text{dup}}$  then
26:            $\mathcal{P} \leftarrow \mathcal{P} \cup \{z\}$ ;  $n \leftarrow n + 1$ .
27:         end if
28:       end if
29:     end for
30:   end for
31:   Append  $n$  to  $\mathcal{N}$ .
32:   // Component 3: Adaptive Broadcast Scheduling (Sec. 3.3)
33:   if  $|\mathcal{N}| \equiv 0 \pmod{W}$  then
34:      $j \leftarrow |\mathcal{N}|/W$ ;  $g_j \leftarrow \frac{1}{W} \sum_{s=(j-1)W+1}^{jW} \mathcal{N}[s]$ .
35:     if  $g_{\text{ref}} = \text{NONE}$  then
36:        $g_{\text{ref}} \leftarrow g_j$ .
37:     else
38:        $r_j \leftarrow g_j / (g_{\text{ref}} + \epsilon)$ .
39:       if mode = PROBE and  $r_j < \tau_{\text{start}}$  then
40:         mode  $\leftarrow$  BROADCAST.
41:       else if mode = BROADCAST and  $r_j < \tau_{\text{stop}}$  then
42:         mode  $\leftarrow$  FREERUN.
43:       end if
44:     end if
45:   end if
46:   if mode = FREERUN then
47:     for each unfinished branch  $i \in [K]$  in parallel do
48:       Continue decoding branch  $i$  until EOS or  $L_{\max}$  is reached, without further synchronization.
49:     end for
50:     break
51:   end if
52: end while
53: return  $\{h_i\}_{i=1}^K$ .
```

B Details of Preliminary Studies

B.1 Information Statistics during Parallel Search

We provide details for the information-statistics experiment in Figure 1. The experiment is conducted on HMMT24–25 using QWEN3-30B-A3B-THINKING-2507 with 64 parallel reasoning branches, where each search step corresponds to a 1024-token generation chunk. At each step, we use the same information-extraction prompt as CPT, detailed in Appendix I, to extract compact information units from branch trajectories. For each problem, we maintain a running analysis pool and compare each extracted candidate with existing pool entries using the same embedding-based semantic de-duplication protocol as CPT: if its maximum similarity to existing entries is below the de-duplication threshold, it is admitted into the pool and counted as newly discovered information for the current step; otherwise, it is counted as duplicate information for the current step. Since one search step consumes 64×1024 generated tokens, we normalize the raw information counts and report the number of information units per 10,240 generated tokens, i.e., multiplying each per-step count by $10,240/(64 \times 1024)$.

B.2 Offline Information Injection

We provide details for the offline information-injection experiment in Table 1, which tests whether information originally scattered across different reasoning branches is decision-useful once shared. For each problem, we first collect parallel reasoning trajectories, use the same information-extraction prompt as CPT (Appendix I) to extract compact information units, and merge them into a deduplicated information pool using the same semantic de-duplication protocol. Given an injection ratio $\rho \in \{0, 20, 40, 60, 80, 100\}$, we randomly sample $\rho\%$ of the deduplicated pool entries and inject them into the initial shared-information section of the prompt, where 0% denotes no injected information and 100% denotes using the full pool. We then run standard parallel sampling with 64 branches on HMMT24–25 using QWEN3-4B-THINKING-2507, without online search-time extraction, pool updating, or broadcasting. All branches receive the same injected context at the beginning, and we report Pass@1, generated tokens, and wall-clock latency. All injected information is extracted from model-generated trajectories,

without using gold answers, external verifiers, or human annotations. We repeat each setting for 8 independent runs and report the averaged results.

C Analysis of Generated Tokens and FLOPs

C.1 Additional Accuracy–Cost Results

We further provide supplementary cost-efficiency results under generated-token and FLOPs metrics. The token-based comparison (Figure 4) complements our latency evaluation by measuring the amount of generated computation. CPT generally achieves a stronger accuracy–tokens trade-off, suggesting that search-time information sharing reduces redundant decoding caused by repeated rediscovery across parallel branches. In contrast, the FLOPs comparison (Figure 5) reflects the additional prefilling computation introduced by prompt-level broadcasts. Since updating the shared-information section changes the input context, standard decoding implementations may need to recompute the prefix, making CPT less favorable under a pure FLOPs budget. These results provide a fuller view of the computational trade-offs of CPT: it improves latency-aware and token-efficient parallel exploration, while FLOPs efficiency remains an important direction for future cache-aware implementations.

C.2 FLOPs Estimation

To explicitly quantify the computational efficiency of different inference strategies, we estimate the Floating Point Operations (FLOPs) following the methodology used in the NVIDIA NeMo framework (nem, 2025).

Unified FLOPs Estimator. We decompose an inference process into a set of model requests, denoted by $\mathcal{R}$. For each request $r \in \mathcal{R}$, we define S_r as the effective linear token term, which is used to estimate the cost of linear projections, MLP layers, and vocabulary projection. We further define Q_r as the effective quadratic attention term, which captures the cost of causal self-attention. The aggregated statistics are:

$$S = \sum_{r \in \mathcal{R}} S_r, \quad (1)$$

$$Q = \sum_{r \in \mathcal{R}} Q_r. \quad (2)$$

Let h denote the hidden size, N_ℓ the number of Transformer layers, A the number of attention

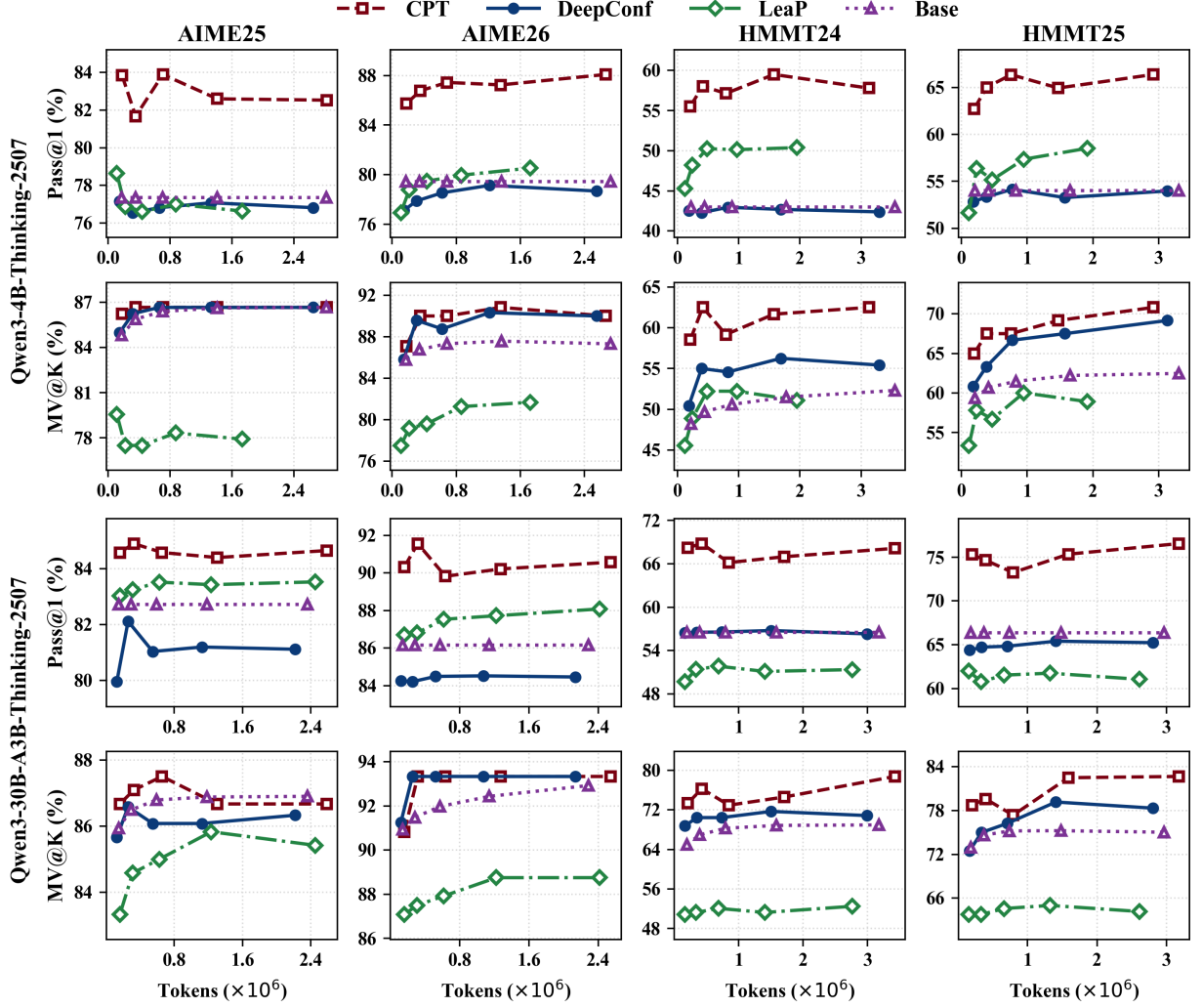


Figure 4: Accuracy–Tokens comparison across models and benchmarks under different rollout budgets.

heads, G the number of key-value heads, V the vocabulary size, and $d_{\text{ff}}^{\text{active}}$ the active FFN intermediate dimension. For dense models, $d_{\text{ff}}^{\text{active}}$ is the standard FFN intermediate dimension; for MoE models, it is multiplied by the number of activated experts per token.

The total FLOPs are decomposed into attention, MLP, and vocabulary projection costs:

$$\text{FLOPs}_{\text{attn}} = 2N_{\ell}h^2 \left[\left(2\frac{G}{A} + 2 \right) S + \frac{Q}{h} \right], \quad (3)$$

$$\text{FLOPs}_{\text{mlp}} = 2N_{\ell}h (3d_{\text{ff}}^{\text{active}}) S, \quad (4)$$

$$\text{FLOPs}_{\text{vocab}} = 2hVS. \quad (5)$$

Therefore, the total computational cost is:

$$\text{FLOPs}_{\text{total}} = \text{FLOPs}_{\text{attn}} + \text{FLOPs}_{\text{mlp}} + \text{FLOPs}_{\text{vocab}}. \quad (6)$$

Full-prefill Generation. For a standard independent generation request, let p_r denote the number

of prompt tokens and o_r denote the number of generated output tokens. Since each request processes the full prompt before autoregressive decoding, we define:

$$S_r = p_r + o_r, \quad (7)$$

$$Q_r = (p_r + o_r)^2. \quad (8)$$

CPT. CPT consists of two types of computation: normal reasoning generation and search-information extraction. Let $\mathcal{R}_{\text{gen}}$ denote the set of normal reasoning requests, and $\mathcal{R}_{\text{ext}}$ denote the set of information extraction requests.

For a normal reasoning request $r \in \mathcal{R}_{\text{gen}}$, let L_r be its effective sequence length. We use:

$$S_r = L_r, \quad (9)$$

$$Q_r = L_r^2. \quad (10)$$

For an information extraction request $r \in \mathcal{R}_{\text{ext}}$, we estimate its cost under a cached-decoding assumption. Let p_r be the length of the extraction

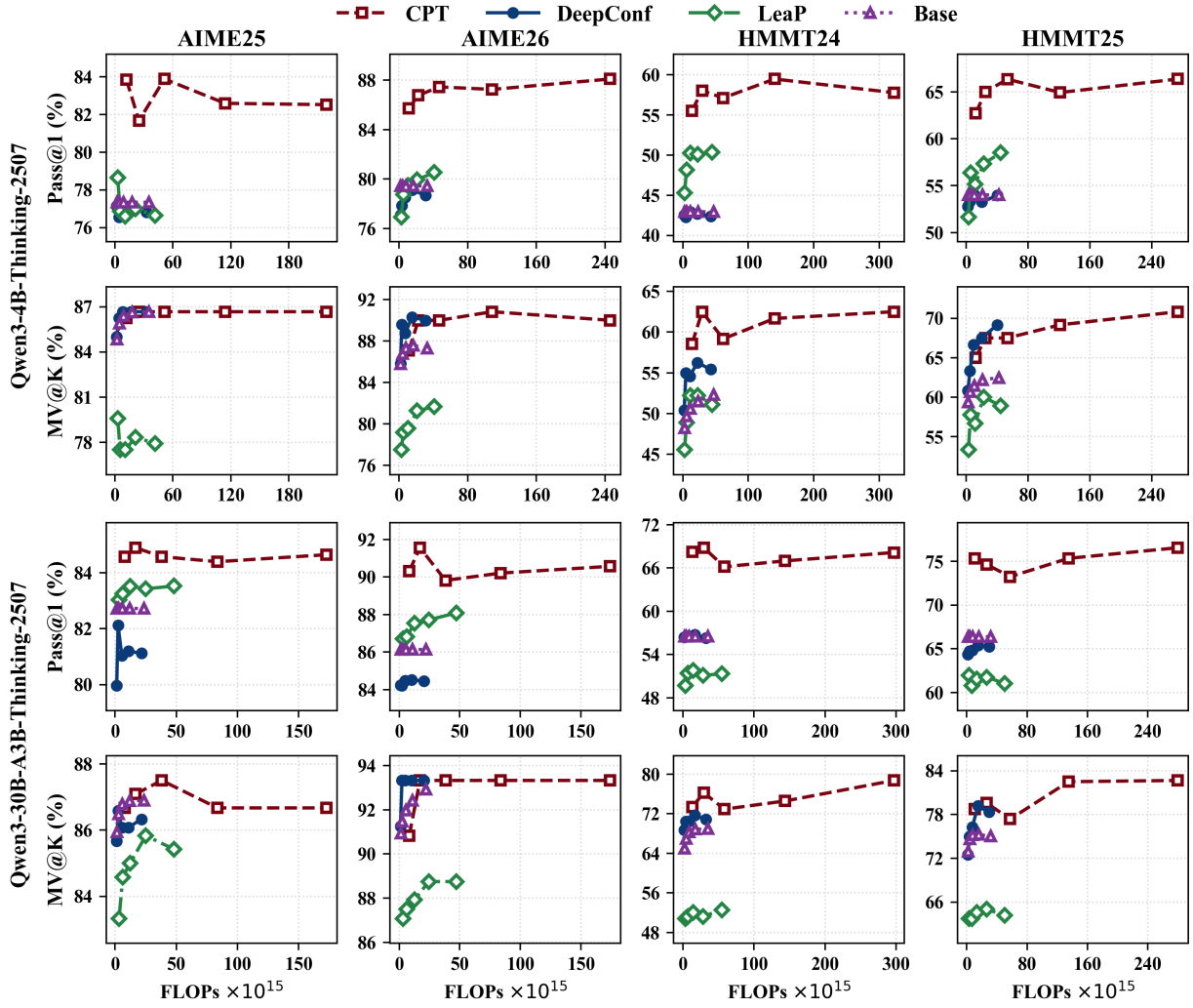


Figure 5: Accuracy–FLOPs comparison across models and benchmarks under different rollout budgets. See Appendix C.2 for details on FLOPs estimation.

context and o_r be the number of tokens generated during extraction. Since the context has already been cached, the linear term only counts newly generated tokens:

$$S_r = o_r. \quad (11)$$

The attention term is:

$$Q_r = 2p_r o_r + o_r^2. \quad (12)$$

Thus, the overall token-dependent statistics for CPT are:

$$S_{\text{CPT}} = \sum_{r \in \mathcal{R}_{\text{gen}}} L_r + \sum_{r \in \mathcal{R}_{\text{ext}}} o_r, \quad (13)$$

$$Q_{\text{CPT}} = \sum_{r \in \mathcal{R}_{\text{gen}}} L_r^2 + \sum_{r \in \mathcal{R}_{\text{ext}}} (2p_r o_r + o_r^2). \quad (14)$$

LeaP. LeaP performs inference through multiple consecutive segments, including normal reasoning,

intermediate summarization, comment insertion, and final-answer generation. Since these segments are generated sequentially within the same trajectory, we estimate its cost using an incremental KV-cache formulation.

For segment r , let C_r denote the number of tokens cached before this segment, and let

$$x_r = p_r + o_r \quad (15)$$

denote the number of newly processed tokens in the current segment, where p_r is the newly inserted prompt length and o_r is the generated output length. The effective statistics are:

$$S_r = x_r, \quad (16)$$

$$Q_r = x_r(2C_r + x_r). \quad (17)$$

Therefore, the overall statistics for LeaP are:

$$S_{\text{LeaP}} = \sum_{r \in \mathcal{R}_{\text{LeaP}}} x_r, \quad (18)$$

$$Q_{\text{LeaP}} = \sum_{r \in \mathcal{R}_{\text{LeaP}}} x_r (2C_r + x_r). \quad (19)$$

DeepConf. DeepConf is treated as a set of independent sampled completions. For problem q , let p_q denote the prompt length after applying the chat template, and let $o_{q,i}$ denote the generated length of the i -th completion. Since each completion is sampled independently, each one is counted as a full-prefill request:

$$S_{q,i} = p_q + o_{q,i}, \quad (20)$$

$$Q_{q,i} = (p_q + o_{q,i})^2. \quad (21)$$

The overall statistics for DeepConf are:

$$S_{\text{DeepConf}} = \sum_q \sum_i (p_q + o_{q,i}), \quad (22)$$

$$Q_{\text{DeepConf}} = \sum_q \sum_i (p_q + o_{q,i})^2. \quad (23)$$

Finally, for each method, the corresponding S and Q are substituted into the unified FLOPs estimator to obtain the total computational cost.

D Mechanism Controls and Component Ablations

To isolate whether CPT benefits from query-relevant cross-branch sharing rather than from the blackboard format or extra prompt context alone, we conduct a matched control suite on HMMT25 using QWEN3-4B-THINKING-2507 with $K = 64$. The controls keep the main experimental configuration fixed while changing one aspect of information construction or delivery: *Empty blackboard* retains the blackboard serialization but provides no shared items; *Cross-problem blackboard* supplies information collected for other problems rather than the current query; *Broadcast without deduplication* removes the semantic filter; *Fixed broadcast schedule* replaces the adaptive PROBE-BROADCAST-FREERUN schedule; *Extraction without cross-branch broadcast* retains information extraction but prevents extracted notes from being disseminated across branches; and *No blackboard instruction* removes the explicit instruction for handling the blackboard while retaining the broadcast content. We do not alter any other setting for these controls.

As shown in Table 6, the controls form a clear hierarchy. Empty and cross-problem blackboards, together with extraction without cross-branch broadcast, occupy the lowest accuracy tier despite preserving either the serialization format, additional context, or extraction compute. In particular, the extraction-only control performs worst on Pass@1, showing that producing useful notes is insufficient unless those notes become available to peer branches. The matched comparison therefore rules out blackboard formatting and generic context augmentation as primary explanations: the shared content must be both query-relevant and disseminated across active branches. The remaining controls separate the roles of pool quality and scheduling. Removing semantic de-duplication simultaneously worsens accuracy and increases generated tokens, consistent with redundant pool entries reintroducing the search work that sharing is intended to avoid. Replacing the adaptive schedule preserves much more of the accuracy but produces a revealing token-latency inversion: it generates fewer tokens yet takes longer, indicating that synchronization timing affects the wall-clock critical path independently of decoding volume. Thus, de-duplication mainly improves the informational efficiency of the pool, whereas adaptive scheduling controls when the cost of sharing is worthwhile. Finally, removing the explicit blackboard instruction produces the smallest and mixed accuracy changes among all controls, with modest cost increases. This pattern suggests that CPT’s main benefit comes from the shared branch information and its delivery mechanism rather than from a particular instruction wording.

E Case Analysis

We provide a paired qualitative case study in Figure 7 to illustrate the “Share More, Search Less” behavior of CPT. The example is taken from a CPT run on a 13th-root-of-unity problem and a no-broadcast continuation initialized from the exact same prefix immediately before the Round 9 broadcast point. The comparison should therefore be read as a same-prefix broadcast intervention: before this point the two contexts are identical; afterward, only the CPT branch receives the shared pool entries. Both branches have already reduced the magnitude condition to a cosine/difference constraint, and then face the same remaining search over candidate exponent sets. At Round 9, the

Variant	Pass@1 (%)	MV@64 (%)	Tokens ($\times 10^6$)	Latency (s)
CPT (matched reference)	66.79	67.33	1.47	402.82
Empty blackboard	52.56	61.46	1.675	374.11
Cross-problem blackboard	52.44	59.83	1.508	564.16
Broadcast without deduplication	56.99	64.37	1.632	393.58
Fixed broadcast schedule	65.42	66.67	1.267	470.32
Extraction without cross-branch broadcast	52.06	59.58	1.597	449.81
No blackboard instruction	66.04	68.33	1.518	415.72

Table 6: Matched mechanism controls on HMMT25 using QWEN3-4B-THINKING-2507 with $K = 64$. Each row reports the mean of eight independent runs; generated tokens are shown in millions and latency in seconds.

CPT branch receives selected pool entries through the shared-information section of the input context; these entries are automatically extracted from model-generated branches and do not use the gold answer, an external verifier, or human annotation. To make the comparison precise, Figure 7 focuses only on the two broadcast notes that are later rediscovered by the no-broadcast trajectory. Note #47 states that the six pairwise positive differences should be exactly $\{1, 2, 3, 4, 5, 6\}$, and note #53 identifies the concrete minimal instance $\{1, 2, 5, 7\}$. Together, these notes provide the difference-set certificate

$$\{1, 2, 5, 7\} \rightsquigarrow \{1, 2, 3, 4, 5, 6\},$$

which gives

$$D = \sum_{k=1}^6 \cos(2\pi k/13) = -1/2,$$

$$|S|^2 = 4 + 2D = 3.$$

The broadcast branch receives this structural certificate from the pool and uses it immediately. The matched no-broadcast trajectory starts from the same prefix but receives no pool information. It eventually reaches the same solution, but only after a substantially longer private continuation that rediscovers the same difference-set certificate through enumeration.

Why this case is useful. The case illustrates that CPT’s Shared Information Pool provides reusable decision information rather than merely exposing final-answer hints. The decisive object is not the final number itself, but the intermediate difference-set certificate that makes the modulus condition checkable. With broadcast, this certificate is available before the branch would otherwise have to find it. Without broadcast, the branch can still discover the same certificate on its own and finish the problem, but it spends a much longer private continuation doing the same search work.

F Generalization beyond Mathematical Domains

To test whether search-time information sharing transfers beyond contest mathematics, we evaluate CPT on GPQA-Diamond (Rein et al., 2024), a graduate-level scientific question-answering benchmark, and LiveCodeBench-v6 (Jain et al.), a code-generation benchmark. We follow the rollout budgets, latency accounting, baselines, and repeated-run protocol of the main experiments, and report results for both QWEN3-4B-THINKING-2507 and QWEN3-30B-A3B-THINKING-2507.

On GPQA-Diamond, CPT ranks first at matched rollout budget for both Pass@1 and MV@K in every model-metric-budget comparison, while also remaining faster than the interaction-based LeaP baseline throughout the sweep. The consistency across both individual-branch and majority-vote accuracy shows that sharing improves branch quality without weakening aggregate decision quality; the result is a systematic frontier shift rather than an isolated best operating point. LiveCodeBench-v6 presents a more conditional pattern. CPT ranks first at four of the five rollout budgets on each backbone, with both exceptions occurring in the low-budget regime. Accuracy is also non-monotonic as the budget grows, indicating that code generation does not benefit uniformly from additional parallel exploration or shared context. Thus, the cross-domain evidence is strongest for scientific reasoning, while code generation supports a narrower claim: CPT improves the upper accuracy-latency envelope, but its benefit depends more strongly on the operating point.

G Complete Accuracy-Latency Results

For completeness, Tables 7–9 and Tables 10–11 jointly report every point underlying the accuracy-latency plots in Figures 3 and 6. Both sets follow the common protocol in Section 4.1; the tables

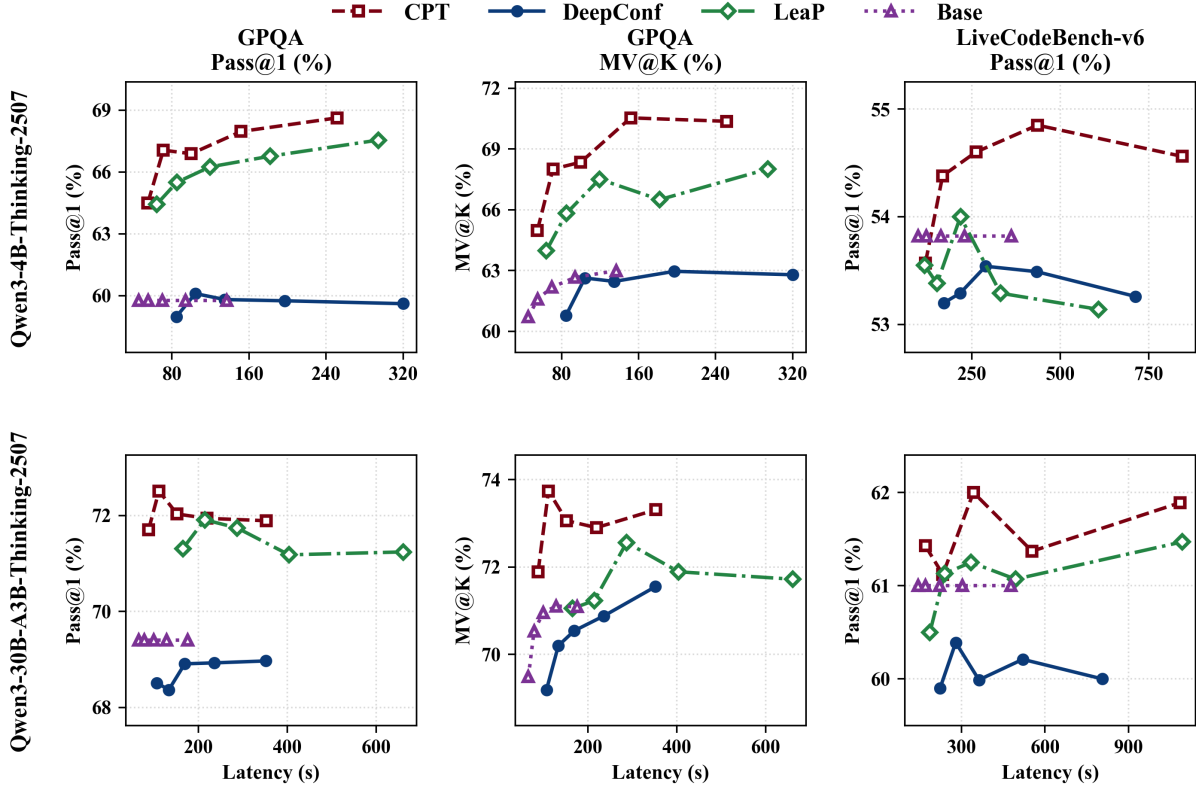


Figure 6: Accuracy-latency comparison on GPQA-Diamond and LiveCodeBench-v6 across rollout budgets. We report Pass@1 and MV@K for GPQA-Diamond and Pass@1 for LiveCodeBench-v6.

retain the run-to-run variation so that the aggregate patterns discussed below can be checked against individual operating points.

Across the mathematical benchmarks, CPT leads or ties in 102/120 matched accuracy comparisons when Pass@1 and MV@K are considered jointly. The cost pattern is more consistent across backbones: CPT is faster than DeepConf in every matched configuration, with an average latency reduction of 39.4%. These two observations separate a broadly stable latency advantage from an accuracy gain whose pointwise consistency depends on the policy model.

The cross-domain tables reveal a parallel task effect. GPQA-Diamond extends the broad dominance observed on the Qwen3 mathematical benchmarks, whereas LiveCodeBench-v6 concentrates its gains away from the smallest budgets and exhibits a non-monotonic response to additional rollouts. Taken together, the complete results support the accuracy-latency-envelope interpretation in Section 4.2 and the cross-domain conclusions in Appendix F, while locating the main exceptions by model family and task rather than obscuring them behind selected cell values.

H Theory: Collaborative Sharing Improves Global Decision Information

We provide an information-theoretic analysis of collaborative parallel thinking. The analysis abstracts away implementation details such as when to broadcast or how many items to broadcast, and focuses on the core question: how does sharing intermediate search information affect the amount of *global decision information* obtained from parallel exploration?

Our main observation is that parallel reasoning paths may produce substantial path-wise information, but the information useful to the final decision is not simply the sum of path-wise information across paths. Different paths may rediscover the same conclusions, constraints, or failure patterns, causing overlapping information to be counted multiple times in isolated exploration. Collaborative sharing can improve test-time scaling efficiency by pooling path-private information into a shared context, reducing redundant local information, and converting it into globally useful decision information.

Benchmark	Method	Metric	Rollout budget K				
			8	16	32	64	128
AIME25	CPT	Pass@1	83.85±1.42	81.67±1.82	83.88±1.03	82.59±0.99	82.52±0.69
		MV@ K	86.25±0.83	86.67±0.00	86.67±0.00	86.67±0.00	86.67±0.00
		Latency	134.9±2.2	172.4±2.1	239.2±4.2	361.7±6.9	650.9±10.7
	DeepConf	Pass@1	77.15±1.07	76.56±1.32	76.82±0.84	77.08±0.66	76.82±0.69
		MV@ K	85.00±1.78	86.25±1.18	86.67±0.00	86.67±0.00	86.67±0.00
		Latency	227.6±3.9	289.0±5.4	385.9±4.0	585.8±4.8	971.5±3.9
	LeaP	Pass@1	78.65±3.51	76.87±1.57	76.61±1.54	77.00±1.62	76.64±1.40
		MV@ K	79.58±3.30	77.50±1.54	77.50±1.54	78.33±2.52	77.92±1.73
		Latency	92.5±1.4	122.0±2.6	171.6±2.7	260.0±4.0	432.9±6.2
	Base	Pass@1	77.35±1.37	77.35±1.61	77.35±0.90	77.35±0.60	77.35±0.27
		MV@ K	84.83±3.09	85.89±1.10	86.40±1.10	86.63±0.00	86.67±0.00
		Latency	128.8±2.0	161.4±2.2	220.3±2.4	317.5±2.6	497.9±1.9
AIME26	CPT	Pass@1	85.73±2.13	86.77±1.37	87.45±0.48	87.25±0.92	88.09±0.24
		MV@ K	87.08±2.10	90.00±0.00	90.00±0.00	90.83±1.67	90.00±0.00
		Latency	133.0±3.2	172.3±2.5	237.4±1.6	352.7±4.2	671.0±6.0
	DeepConf	Pass@1	77.10±1.64	77.87±1.12	78.53±0.97	79.13±1.55	78.67±0.52
		MV@ K	85.83±2.95	89.58±1.18	88.75±1.73	90.32±0.92	90.00±0.00
		Latency	218.2±6.3	277.4±4.0	371.5±5.0	560.2±3.9	936.6±3.8
	LeaP	Pass@1	76.93±1.81	78.76±4.14	79.47±2.67	79.95±2.30	80.53±1.64
		MV@ K	77.50±2.36	79.17±4.63	79.58±2.78	81.25±2.48	81.67±2.52
		Latency	91.1±1.9	119.6±1.8	168.2±2.2	252.7±4.9	414.9±4.7
	Base	Pass@1	79.45±1.09	79.45±1.05	79.45±0.68	79.45±0.32	79.45±0.25
		MV@ K	85.79±1.61	86.76±2.76	87.32±1.10	87.57±1.61	87.31±1.61
		Latency	128.1±3.4	154.5±2.4	210.5±3.0	307.5±2.2	477.3±1.9
HMMT24	CPT	Pass@1	55.52±3.69	58.02±2.37	57.11±4.22	59.47±1.38	57.77±4.02
		MV@ K	58.54±3.48	62.50±3.19	59.17±3.19	61.67±1.92	62.50±3.19
		Latency	166.7±3.3	216.2±3.9	298.2±8.2	452.1±4.8	846.5±29.7
	DeepConf	Pass@1	42.54±1.04	42.29±1.38	42.95±0.91	42.69±0.60	42.38±0.46
		MV@ K	50.42±4.52	55.00±2.52	54.58±2.48	56.25±2.78	55.42±1.73
		Latency	283.3±4.5	354.8±7.3	483.2±4.8	745.3±5.8	1246.5±9.8
	LeaP	Pass@1	45.27±2.04	48.18±1.97	50.24±1.04	50.12±1.94	50.37±1.44
		MV@ K	45.56±1.92	48.89±1.92	52.22±1.92	52.22±1.92	51.11±1.92
		Latency	100.9±1.2	133.8±1.9	190.8±4.1	285.8±3.1	479.6±6.5
	Base	Pass@1	42.96±2.20	42.96±1.14	42.96±0.45	42.96±0.27	42.96±0.19
		MV@ K	48.24±2.60	49.70±3.51	50.60±3.63	51.49±3.23	52.31±1.67
		Latency	159.6±2.4	199.4±3.7	279.3±3.0	410.2±3.3	676.0±5.3
HMMT25	CPT	Pass@1	62.71±3.15	65.00±2.27	66.35±0.74	64.93±1.54	66.37±1.58
		MV@ K	65.00±3.33	67.50±1.67	67.50±1.67	69.17±1.67	70.83±1.67
		Latency	148.9±3.8	193.5±4.2	270.9±5.4	398.5±7.0	732.7±9.7
	DeepConf	Pass@1	52.81±2.44	53.33±1.33	54.14±0.95	53.26±0.50	53.96±0.53
		MV@ K	60.83±5.84	63.33±6.17	66.67±2.52	67.50±2.36	69.17±1.54
		Latency	272.9±3.1	343.5±2.3	458.4±2.6	702.6±4.8	1169.9±11.5
	LeaP	Pass@1	51.66±2.72	56.38±2.52	55.16±5.68	57.34±3.25	58.51±1.15
		MV@ K	53.33±3.33	57.78±1.92	56.67±6.67	60.00±3.33	58.89±1.92
		Latency	96.0±2.1	129.3±2.8	184.9±4.3	280.0±3.8	468.5±13.0
	Base	Pass@1	54.02±2.00	54.02±1.07	54.02±1.01	54.02±0.54	54.02±0.24
		MV@ K	59.38±5.38	60.67±4.00	61.46±3.33	62.20±2.76	62.46±1.44
		Latency	151.6±1.6	193.2±1.2	261.6±1.6	381.1±2.7	610.6±6.1

Table 7: Complete accuracy–latency results for QWEN3-4B-THINKING-2507 underlying Figure 3. Pass@1 and MV@ K are percentages; latency is in seconds. Each entry is the mean ± reported run-to-run variation over eight independent runs. Accuracy is rounded to two decimal places and latency to one decimal place.

Benchmark	Method	Metric	Rollout budget K				
			8	16	32	64	128
AIME25	CPT	Pass@1	84.58±0.92	84.90±0.69	84.58±0.86	84.40±0.47	84.65±0.82
		MV@ K	86.67±0.00	87.09±0.84	87.50±1.67	86.67±0.00	86.67±0.00
		Latency	190.5±4.7	243.0±3.8	334.0±3.8	511.4±8.2	906.7±28.9
	DeepConf	Pass@1	79.96±1.38	82.12±1.89	81.04±0.51	81.20±0.80	81.12±0.41
		MV@ K	85.67±2.52	86.58±3.30	86.08±2.14	86.08±2.14	86.33±1.78
		Latency	280.4±5.9	355.7±8.6	468.6±5.6	669.6±4.6	1041.0±11.8
	LeaP	Pass@1	83.03±1.06	83.25±1.81	83.52±2.13	83.43±1.48	83.53±1.32
		MV@ K	83.33±1.78	84.58±1.73	85.00±2.52	85.83±1.54	85.42±1.73
		Latency	261.1±6.1	334.8±12.4	450.0±16.0	633.7±23.1	1401.6±41.3
	Base	Pass@1	82.72±0.83	82.72±0.85	82.72±0.66	82.72±0.51	82.72±0.27
		MV@ K	85.94±2.20	86.49±2.32	86.79±1.61	86.88±1.67	86.90±1.67
		Latency	174.2±3.3	223.2±5.8	277.7±3.5	394.3±2.7	609.7±6.6
AIME26	CPT	Pass@1	90.31±1.29	91.56±0.69	89.82±0.23	90.21±0.88	90.57±0.91
		MV@ K	90.83±1.67	93.33±0.00	93.33±0.00	93.33±0.00	93.33±0.00
		Latency	190.3±3.2	232.8±4.3	319.0±4.6	490.4±6.6	896.0±7.8
	DeepConf	Pass@1	84.25±1.26	84.21±1.66	84.49±0.94	84.52±0.38	84.46±0.50
		MV@ K	91.25±1.73	93.33±0.00	93.33±0.00	93.33±0.00	93.33±0.00
		Latency	274.5±9.1	348.2±7.3	453.4±8.8	645.0±8.2	993.5±9.9
	LeaP	Pass@1	86.71±2.90	86.81±1.91	87.53±1.03	87.72±0.84	88.08±1.13
		MV@ K	87.08±3.30	87.50±2.36	87.92±1.73	88.75±1.73	88.75±1.73
		Latency	250.6±6.2	319.8±3.8	427.9±6.0	601.5±8.0	1339.1±25.7
	Base	Pass@1	86.15±0.77	86.15±0.75	86.15±0.35	86.15±0.40	86.15±0.28
		MV@ K	90.94±1.44	91.48±1.44	91.97±1.44	92.43±1.44	92.92±1.44
		Latency	178.8±5.5	210.6±4.9	278.4±5.7	382.1±4.9	565.7±5.5
HMMT24	CPT	Pass@1	68.23±2.73	68.80±1.76	66.17±0.97	66.98±2.77	68.14±3.52
		MV@ K	73.33±2.72	76.25±0.83	72.92±2.85	74.58±1.59	78.73±3.54
		Latency	263.9±6.5	338.4±5.2	462.8±11.8	725.9±14.2	1342.6±31.4
	DeepConf	Pass@1	56.43±2.94	56.52±1.27	56.56±1.23	56.76±1.13	56.30±1.00
		MV@ K	68.75±3.54	70.42±3.75	70.42±2.78	71.67±1.78	70.83±1.54
		Latency	388.4±10.6	487.6±6.2	637.7±5.9	916.0±6.3	1453.9±9.2
	LeaP	Pass@1	49.68±3.17	51.38±2.72	51.82±1.39	51.10±1.85	51.35±1.56
		MV@ K	50.83±3.45	51.25±3.54	52.08±1.73	51.25±2.48	52.50±1.54
		Latency	254.5±5.5	333.3±7.2	458.7±7.6	670.9±6.0	1665.6±24.1
	Base	Pass@1	56.52±1.59	56.52±1.35	56.52±0.89	56.52±0.57	56.52±0.27
		MV@ K	64.96±2.89	66.93±2.60	68.27±3.31	68.84±1.44	68.94±2.00
		Latency	236.6±6.2	293.0±3.4	386.5±3.9	540.4±3.9	891.6±5.6
HMMT25	CPT	Pass@1	75.31±0.52	74.64±2.50	73.23±1.70	75.33±2.38	76.54±1.10
		MV@ K	78.75±2.09	79.59±3.44	77.37±0.84	82.50±3.19	82.67±1.19
		Latency	243.2±5.9	306.9±5.5	425.7±8.7	689.7±11.2	1284.7±18.8
	DeepConf	Pass@1	64.37±2.10	64.72±1.13	64.83±0.84	65.40±1.07	65.23±0.55
		MV@ K	72.50±3.45	75.00±2.52	76.25±4.52	79.17±2.36	78.33±1.78
		Latency	367.1±7.7	457.3±6.9	598.2±6.5	862.0±3.7	1373.3±12.1
	LeaP	Pass@1	61.98±3.20	60.78±2.45	61.53±3.05	61.75±2.78	61.05±2.13
		MV@ K	63.75±3.30	63.75±3.30	64.58±2.48	65.00±3.56	64.17±4.27
		Latency	240.1±5.5	326.3±5.8	446.0±5.1	639.1±8.9	1519.1±20.4
	Base	Pass@1	66.37±1.10	66.37±1.37	66.37±0.82	66.37±0.58	66.37±0.37
		MV@ K	72.96±3.09	74.62±3.89	75.21±2.60	75.24±4.23	75.03±3.73
		Latency	221.4±4.5	271.4±3.8	365.4±4.3	510.1±2.3	826.6±7.4

Table 8: Complete accuracy–latency results for QWEN3-30B-A3B-THINKING-2507 underlying Figure 3. Pass@1 and MV@ K are percentages; latency is in seconds. Each entry is the mean $\pm$ reported run-to-run variation over eight independent runs. Accuracy is rounded to two decimal places and latency to one decimal place.

Benchmark	Method	Metric	Rollout budget K				
			8	16	32	64	128
AIME25	CPT	Pass@1	54.48±2.60	66.82±0.69	71.38±1.48	69.11±1.72	71.52±1.76
		MV@ K	80.93±3.09	83.44±2.64	85.44±2.92	84.17±1.67	84.17±1.67
		Latency	176.7±4.3	194.6±6.9	231.4±6.6	317.2±9.4	493.2±30.8
	DeepConf	Pass@1	38.40±5.77	48.45±2.33	58.73±2.37	62.44±2.40	61.46±2.71
		MV@ K	46.67±9.43	71.67±7.77	82.58±4.69	84.50±3.88	85.92±3.05
		Latency	272.1±10.1	389.2±2.8	510.3±6.7	914.6±26.8	1870.3±50.9
	LeaP	Pass@1	41.53±13.61	64.09±2.95	69.57±1.69	70.99±2.57	69.62±1.31
		MV@ K	50.42±19.31	77.50±3.88	79.58±2.78	79.58±2.78	75.83±2.36
		Latency	201.0±12.7	208.1±3.6	240.6±2.0	296.8±4.4	391.3±4.4
	Base	Pass@1	65.26±1.37	64.82±1.29	65.29±0.52	64.90±0.52	64.99±0.33
		MV@ K	75.00±1.67	76.25±1.10	77.08±2.60	76.25±1.10	76.67±0.00
		Latency	150.6±6.6	156.7±4.1	176.5±3.3	213.9±4.8	262.2±8.7
AIME26	CPT	Pass@1	60.31±2.77	69.32±1.30	75.98±1.85	76.34±2.07	77.36±0.74
		MV@ K	87.44±1.11	87.91±1.59	90.83±1.67	90.00±0.01	92.91±0.84
		Latency	169.1±3.9	188.9±5.1	227.2±9.0	316.2±16.8	488.0±17.8
	DeepConf	Pass@1	37.68±3.05	47.39±2.79	61.72±1.52	65.26±2.22	65.87±2.18
		MV@ K	45.83±10.50	67.50±6.84	88.83±2.95	88.25±2.48	91.18±2.56
		Latency	272.8±5.8	367.5±5.8	510.7±11.7	870.7±26.4	1793.3±59.3
	LeaP	Pass@1	54.06±5.13	69.18±3.87	75.86±1.18	75.57±2.71	76.48±0.91
		MV@ K	68.75±7.33	81.25±4.34	87.92±1.73	87.08±3.75	87.92±1.94
		Latency	187.3±0.9	205.5±1.2	233.2±2.0	288.5±2.0	377.1±4.1
	Base	Pass@1	67.34±2.26	67.84±1.61	68.37±0.65	68.44±0.89	68.17±0.31
		MV@ K	81.67±3.73	85.42±2.32	86.25±2.00	86.67±0.00	86.67±0.00
		Latency	148.1±2.0	156.4±2.6	177.1±4.3	209.7±6.3	257.1±6.1
HMMT24	CPT	Pass@1	32.81±3.78	37.73±2.02	40.79±1.55	40.62±0.98	41.19±0.92
		MV@ K	44.44±1.54	53.10±3.79	53.80±4.33	58.33±1.92	61.68±4.07
		Latency	181.1±3.3	207.2±4.9	265.2±10.2	377.8±13.9	603.2±32.5
	DeepConf	Pass@1	20.47±4.26	23.88±2.67	35.12±1.75	36.35±0.80	36.59±2.97
		MV@ K	27.50±10.95	47.92±5.62	58.50±4.27	64.50±5.27	65.35±5.58
		Latency	279.7±3.0	382.7±4.9	577.9±13.2	1067.5±32.8	2217.5±65.9
	LeaP	Pass@1	23.34±8.87	36.11±2.97	38.62±3.10	40.07±1.77	39.99±1.10
		MV@ K	28.33±13.21	47.50±4.27	50.42±3.75	53.33±3.56	52.92±3.75
		Latency	204.3±10.9	227.6±1.6	269.5±2.4	356.0±3.8	487.9±6.7
	Base	Pass@1	37.66±2.10	37.73±1.36	38.35±0.66	37.59±0.44	37.92±0.31
		MV@ K	46.67±3.73	45.83±3.23	50.83±2.20	50.83±3.23	48.75±2.32
		Latency	150.8±5.4	161.0±2.5	186.9±4.5	235.8±5.6	293.5±7.9
HMMT25	CPT	Pass@1	42.71±3.81	52.76±3.86	57.58±1.24	57.28±2.44	57.47±1.76
		MV@ K	63.32±2.36	71.85±1.83	75.70±2.24	77.50±1.67	76.25±0.83
		Latency	176.3±5.4	205.8±5.1	259.9±7.1	383.0±13.4	596.6±21.1
	DeepConf	Pass@1	28.97±1.69	35.83±2.26	50.08±2.48	52.54±1.24	53.51±3.25
		MV@ K	41.67±5.63	63.75±6.22	75.58±4.86	78.08±2.48	80.35±4.58
		Latency	285.7±6.5	384.8±8.1	532.6±18.5	961.6±41.1	1989.0±71.9
	LeaP	Pass@1	35.69±4.44	48.46±3.45	52.40±3.09	54.94±2.72	55.09±2.08
		MV@ K	44.58±4.69	58.33±3.98	61.67±5.63	64.58±3.96	62.71±2.51
		Latency	192.6±1.0	220.7±2.1	260.9±3.4	336.6±2.2	460.7±5.3
	Base	Pass@1	56.82±1.89	54.64±1.66	55.12±1.16	55.08±0.45	55.43±0.45
		MV@ K	68.33±3.33	69.58±2.60	67.50±2.76	67.50±2.20	70.00±2.36
		Latency	145.9±2.6	159.2±2.7	184.5±3.7	227.8±4.7	277.6±6.5

Table 9: Complete accuracy–latency results for GPT-OSS-20B underlying Figure 3. Pass@1 and MV@ K are percentages; latency is in seconds. Each entry is the mean ± reported run-to-run variation over eight independent runs. Accuracy is rounded to two decimal places and latency to one decimal place.

Model	Method	Metric	Rollout budget K				
			8	16	32	64	128
Qwen3-4B Thinking-2507	CPT	Pass@1	64.50±0.80	67.06±1.09	66.89±0.26	67.97±0.94	68.63±0.64
		MV@ K	64.98±0.77	68.01±1.77	68.35±0.29	70.54±0.58	70.37±0.77
		Latency	54.9±0.10	71.0±0.6	100.0±1.6	151.8±2.5	251.5±2.1
	DeepConf	Pass@1	58.98±0.50	60.10±0.43	59.81±0.19	59.75±0.31	59.61±0.08
		MV@ K	60.77±1.05	62.63±1.01	62.46±1.05	62.96±0.77	62.79±0.58
		Latency	84.5±0.8	104.3±0.6	134.8±0.2	197.2±1.2	320.0±1.2
	LeaP	Pass@1	64.44±1.63	65.50±1.27	66.25±1.89	66.77±0.34	67.55±0.75
		MV@ K	63.97±1.62	65.82±1.54	67.51±2.49	66.50±0.58	68.01±0.58
		Latency	64.2±0.5	85.0±0.7	119.4±1.1	181.9±1.5	294.1±0.7
	Base	Pass@1	59.76±0.46	59.76±0.43	59.76±0.20	59.76±0.30	59.76±0.08
		MV@ K	60.72±0.99	61.59±1.06	62.17±1.02	62.66±0.83	62.99±0.58
		Latency	45.6±0.4	55.3±0.3	70.0±0.1	94.0±0.6	136.8±0.5
Qwen3-30B-A3B Thinking-2507	CPT	Pass@1	71.70±0.24	72.51±0.75	72.03±0.15	71.94±0.85	71.89±0.67
		MV@ K	71.89±0.29	73.74±1.34	73.06±0.77	72.90±0.29	73.31±1.13
		Latency	88.1±1.0	111.5±0.7	152.2±2.1	219.3±2.6	352.7±4.2
	DeepConf	Pass@1	68.51±0.27	68.37±0.47	68.91±0.30	68.93±0.41	68.97±0.37
		MV@ K	69.19±2.20	70.20±0.51	70.54±0.77	70.88±1.27	71.55±1.05
		Latency	107.0±1.3	133.7±0.5	169.6±1.5	236.1±2.0	352.2±0.7
	LeaP	Pass@1	71.31±1.74	71.91±1.87	71.74±0.70	71.18±0.56	71.24±0.69
		MV@ K	71.05±2.59	71.23±2.53	72.56±1.62	71.89±0.77	71.72±0.51
		Latency	165.2±3.1	214.4±0.3	287.4±0.7	404.2±2.1	660.7±0.7
	Base	Pass@1	69.40±0.25	69.40±0.47	69.40±0.32	69.40±0.40	69.40±0.38
		MV@ K	69.49±2.02	70.53±0.53	70.95±0.73	71.10±1.35	71.09±1.03
		Latency	66.3±0.8	79.0±0.3	99.5±0.9	128.4±1.2	175.8±0.3

Table 10: Complete accuracy–latency results on GPQA-Diamond underlying Figure 6. Pass@1 and MV@ K are percentages; latency is in seconds. Each entry is the mean $\pm$ reported run-to-run variation over eight independent runs. Accuracy is rounded to two decimal places and latency to one decimal place.

Model	Method	Metric	Rollout budget K				
			8	16	32	64	128
Qwen3-4B Thinking-2507	CPT	Pass@1	53.57±0.62	54.38±0.25	54.60±0.86	54.85±0.14	54.56±0.09
		Latency	118.9±1.4	167.5±2.7	261.7±2.8	435.9±5.8	845.3±8.9
	DeepConf	Pass@1	53.20±0.30	53.29±0.27	53.54±0.31	53.49±0.14	53.26±0.18
		Latency	170.7±1.1	217.1±1.0	289.2±1.5	433.7±0.3	712.7±5.8
	LeaP	Pass@1	53.55±2.21	53.38±1.70	54.00±0.29	53.29±0.61	53.14±0.02
		Latency	116.2±0.2	151.6±0.6	217.9±0.7	331.4±0.8	608.0±0.7
	Base	Pass@1	53.82±0.28	53.82±0.27	53.82±0.33	53.82±0.14	53.82±0.19
		Latency	97.7±0.6	121.6±0.5	163.2±0.9	229.6±0.2	361.4±3.0
Qwen3-30B-A3B Thinking-2507	CPT	Pass@1	61.43±0.14	61.12±0.40	62.00±0.02	61.37±0.34	61.89±0.60
		Latency	170.0±1.3	233.1±2.4	343.1±3.7	553.6±5.7	1085.2±15.5
	DeepConf	Pass@1	59.90±0.74	60.39±0.30	59.99±0.23	60.21±0.09	60.00±0.14
		Latency	223.1±1.6	279.4±1.8	362.8±1.7	520.2±2.2	806.2±1.9
	LeaP	Pass@1	60.50±1.00	61.13±0.89	61.25±1.09	61.07±0.07	61.47±0.19
		Latency	186.7±0.8	239.9±1.3	334.2±2.1	494.5±1.4	1093.3±4.9
	Base	Pass@1	61.00±0.70	61.00±0.31	61.00±0.21	61.00±0.09	61.00±0.15
		Latency	143.7±1.0	170.9±1.0	221.6±0.9	302.9±1.4	477.6±1.1

Table 11: Complete accuracy–latency results on LiveCodeBench-v6 underlying Figure 6. Pass@1 is a percentage and latency is in seconds; MV@ K is not evaluated for this benchmark. Each entry is the mean $\pm$ reported run-to-run variation over eight independent runs. Accuracy is rounded to two decimal places and latency to one decimal place.

H.1 Decision information in parallel reasoning

Fix a query q , and let Y denote the target answer under an epistemic distribution over candidate answers. Although the ground-truth answer is deterministic, the solver is uncertain about it before reasoning; our information-theoretic quantities are defined with respect to this uncertainty. At some stage of inference, let S denote the information state already available to the system. Suppose K parallel reasoning paths continue exploration and produce extracted search information

$$Z_{1:K} = (Z_1, \dots, Z_K),$$

where Z_i denotes the information extracted from path i , such as intermediate conclusions, constraints, counterexamples, or failure patterns. We interpret Z_i as residual search information beyond the current state S .

We measure the decision value of information by its mutual information with the target answer. All entropy and mutual-information quantities below are assumed to be finite; the Gaussian example later uses the continuous analogue.

Definition 1 (Pooled global decision information). Given the current information state S , the pooled global decision information gained from the next parallel exploration step is

$$\mathcal{G}(Z_{1:K} \mid S, q) := I(Y; Z_{1:K} \mid S, q).$$

This quantity measures the decision information obtained after pooling the extracted information from all paths into a shared context.

Definition 2 (Aggregate path-wise information). The aggregate path-wise information produced by the K paths is

$$\mathcal{L}(Z_{1:K} \mid S, q) := \sum_{i=1}^K I(Y; Z_i \mid S, q).$$

This quantity counts the decision information produced by each path separately, so overlapping information across paths is counted multiple times.

The quantity $\mathcal{L}$ measures how much decision-relevant information each path appears to produce in isolation. However, it can overestimate the amount of new information available to the global decision process, because different paths may produce overlapping information.

Definition 3 (Redundant local information). We define the redundant local information as

$$\mathcal{R}(Z_{1:K} \mid S, q) := \mathcal{L}(Z_{1:K} \mid S, q) - \mathcal{G}(Z_{1:K} \mid S, q).$$

Intuitively, $\mathcal{R}$ measures the amount of path-wise decision information that is repeatedly counted across isolated paths but does not translate into additional pooled global decision information.

Assumption 4 (Redundancy-dominated extracted information). For the extracted search information considered in this analysis, pooling information across paths does not create more decision information than the sum of its path-wise contributions:

$$\mathcal{G}(Z_{1:K} \mid S, q) \leq \mathcal{L}(Z_{1:K} \mid S, q).$$

Equivalently,

$$\mathcal{R}(Z_{1:K} \mid S, q) \geq 0.$$

This assumption abstracts the setting where extracted information units are self-contained intermediate conclusions, constraints, counterexamples, or failure patterns, and the main inefficiency comes from repeated or overlapping information across paths.

Decision value under log-loss. The mutual-information objective has a standard decision-theoretic interpretation. If the final predictor is evaluated by log-loss, then the optimal expected loss before observing additional information A is

$$H(Y \mid S, q),$$

whereas the optimal expected loss after observing A is

$$H(Y \mid A, S, q).$$

Therefore, the expected improvement from observing A is

$$H(Y \mid S, q) - H(Y \mid A, S, q) = I(Y; A \mid S, q).$$

Thus, maximizing pooled global decision information is equivalent to maximizing the expected log-loss reduction of an optimal Bayesian decision rule.

H.2 From path-wise information to pooled global information

The definitions above imply a simple but important decomposition:

$$\mathcal{L}(Z_{1:K} \mid S, q) = \mathcal{G}(Z_{1:K} \mid S, q) + \mathcal{R}(Z_{1:K} \mid S, q).$$

Equivalently,

$$\mathcal{G}(Z_{1:K} | S, q) = \mathcal{L}(Z_{1:K} | S, q) - \mathcal{R}(Z_{1:K} | S, q).$$

In words, aggregate path-wise information equals pooled global decision information plus redundant local information. This decomposition captures the central inefficiency of independent parallel exploration: even if each path produces useful local information, repeated discoveries across paths can inflate the path-wise information total without increasing the pooled global decision information.

We next relate redundant local information to a standard information-theoretic dependence measure.

Definition 5 (Conditional total correlation). For random variables $Z_{1:K}$ and conditioning variable A , the conditional total correlation is

$$\text{TC}(Z_{1:K} | A) := \sum_{i=1}^K H(Z_i | A) - H(Z_{1:K} | A).$$

It measures statistical dependence among $Z_1, \dots, Z_K$ given A .

Proposition 6 (Redundant information and total correlation). *For any joint distribution over $Y, Z_{1:K}, S$, conditioned on the fixed query q , we have*

$$\begin{aligned} \mathcal{R}(Z_{1:K} | S, q) &= \text{TC}(Z_{1:K} | S, q) \\ &\quad - \text{TC}(Z_{1:K} | Y, S, q). \end{aligned}$$

In particular, under the clean model where $Z_1, \dots, Z_K$ are mutually conditionally independent given Y, S, q , we have

$$\mathcal{R}(Z_{1:K} | S, q) = \text{TC}(Z_{1:K} | S, q) \geq 0.$$

Proof. Let $A = (S, q)$, $B = (Y, S, q)$, and write $\sum_i$ for $\sum_{i=1}^K$. By the definition of $\mathcal{R}$,

$$\begin{aligned} \mathcal{R} &= \sum_i I(Y; Z_i | A) - I(Y; Z_{1:K} | A) \\ &= \sum_i H(Z_i | A) - H(Z_{1:K} | A) \\ &\quad - \sum_i H(Z_i | B) + H(Z_{1:K} | B) \\ &= \text{TC}(Z_{1:K} | A) - \text{TC}(Z_{1:K} | B). \end{aligned}$$

Restoring $A = (S, q)$ and $B = (Y, S, q)$ gives

$$\begin{aligned} \mathcal{R}(Z_{1:K} | S, q) &= \text{TC}(Z_{1:K} | S, q) \\ &\quad - \text{TC}(Z_{1:K} | Y, S, q). \end{aligned}$$

If $Z_1, \dots, Z_K$ are mutually conditionally independent given Y, S, q , then

$$H(Z_{1:K} | Y, S, q) = \sum_{i=1}^K H(Z_i | Y, S, q),$$

so $\text{TC}(Z_{1:K} | Y, S, q) = 0$. Hence

$$\mathcal{R}(Z_{1:K} | S, q) = \text{TC}(Z_{1:K} | S, q) \geq 0,$$

since total correlation is nonnegative.

Proposition 6 shows that, under a clean conditional-independence model, redundant local information is exactly the total correlation among path-level information variables. Thus, when different paths repeatedly discover similar information, the aggregate path-wise information can be large while the pooled global decision information remains limited.

H.3 When does collaborative sharing help?

We now compare independent parallel exploration with collaborative exploration. Let

$$Z_{1:K}^{\text{ind}}$$

denote the information variables produced by independent parallel paths, and let

$$Z_{1:K}^{\text{col}}$$

denote those produced under collaborative sharing. Both are conditioned on the same current state S and query q .

Define

$$\begin{aligned} \mathcal{G}_{\text{ind}} &:= \mathcal{G}(Z_{1:K}^{\text{ind}} | S, q), \\ \mathcal{G}_{\text{col}} &:= \mathcal{G}(Z_{1:K}^{\text{col}} | S, q), \end{aligned}$$

and define $\mathcal{L}_{\text{ind}}, \mathcal{L}_{\text{col}}$ and $\mathcal{R}_{\text{ind}}, \mathcal{R}_{\text{col}}$ analogously.

The following result is a decomposition rather than an unconditional dominance guarantee.

Theorem 7 (Collaborative gain decomposition). *The difference in pooled global decision information between collaborative and independent exploration satisfies*

$$\begin{aligned} \mathcal{G}_{\text{col}} - \mathcal{G}_{\text{ind}} &= (\mathcal{L}_{\text{col}} - \mathcal{L}_{\text{ind}}) \\ &\quad + (\mathcal{R}_{\text{ind}} - \mathcal{R}_{\text{col}}). \end{aligned}$$

Consequently, collaborative sharing improves pooled global decision information whenever

$$\mathcal{R}_{\text{ind}} - \mathcal{R}_{\text{col}} \geq \mathcal{L}_{\text{ind}} - \mathcal{L}_{\text{col}}.$$

In particular, if collaboration reduces redundant local information without reducing aggregate path-wise information, then

$$\mathcal{G}_{\text{col}} \geq \mathcal{G}_{\text{ind}}.$$

Proof. By the path-wise to pooled decomposition,

$$\mathcal{G}_{\text{col}} = \mathcal{L}_{\text{col}} - \mathcal{R}_{\text{col}}, \quad \mathcal{G}_{\text{ind}} = \mathcal{L}_{\text{ind}} - \mathcal{R}_{\text{ind}}.$$

Subtracting the two identities gives

$$\begin{aligned} \mathcal{G}_{\text{col}} - \mathcal{G}_{\text{ind}} &= (\mathcal{L}_{\text{col}} - \mathcal{R}_{\text{col}}) - (\mathcal{L}_{\text{ind}} - \mathcal{R}_{\text{ind}}) \\ &= (\mathcal{L}_{\text{col}} - \mathcal{L}_{\text{ind}}) + (\mathcal{R}_{\text{ind}} - \mathcal{R}_{\text{col}}). \end{aligned}$$

Thus $\mathcal{G}_{\text{col}} \geq \mathcal{G}_{\text{ind}}$ whenever

$$\mathcal{R}_{\text{ind}} - \mathcal{R}_{\text{col}} \geq \mathcal{L}_{\text{ind}} - \mathcal{L}_{\text{col}}.$$

Theorem 7 formalizes the core intuition behind collaborative parallel thinking. Sharing can help not merely by producing more path-wise information, but by reducing redundant local information that prevents path-wise information from becoming pooled global decision information. The theorem also makes clear that sharing is not unconditionally beneficial: if shared context reduces path-level exploration too much, the decrease in aggregate path-wise information may offset the reduction in redundant local information.

H.4 A Gaussian redundancy model

The previous results are distribution-free identities. We now give a stylized Gaussian model to illustrate how cross-path dependence reduces the effective information width of parallel exploration. This model is not intended as a literal model of LLM reasoning; rather, it isolates the effect of correlated path-level information on global decision information.

For simplicity, omit conditioning on S, q . Let

$$Y \sim \mathcal{N}(0, 1)$$

be a scalar latent decision signal. Each path produces a noisy residual information signal

$$Z_i = Y + U + V_i, \quad i = 1, \dots, K,$$

where

$$U \sim \mathcal{N}(0, \sigma_u^2)$$

is a common noise component shared across paths, and

$$V_i \sim \mathcal{N}(0, \sigma_u^2)$$

is path-specific noise. Assume $U, V_1, \dots, V_K$ are mutually independent and independent of Y . Assume $\sigma_u^2 > 0$ and $\sigma_c^2 \geq 0$; the degenerate case $\sigma_u^2 = 0$ is understood by continuity.

The common noise U represents shared uncertainty or common bias induced by repeated derivations, similar failure modes, or correlated search behavior across paths. The path-specific noise V_i represents idiosyncratic uncertainty that can be averaged out across independent paths.

Let

$$\sigma^2 := \sigma_u^2 + \sigma_c^2, \quad \rho := \frac{\sigma_c^2}{\sigma_u^2 + \sigma_c^2}.$$

Then σ^2 is the marginal noise variance of each path, and ρ is the pairwise noise correlation.

Proposition 8 (Effective parallel width under correlated search information). *Under the Gaussian redundancy model,*

$$I(Y; Z_{1:K}) = \frac{1}{2} \log \left(1 + \frac{K}{\sigma_u^2 + K\sigma_c^2} \right).$$

Equivalently,

$$I(Y; Z_{1:K}) = \frac{1}{2} \log \left(1 + \frac{K_{\text{eff}}}{\sigma^2} \right),$$

where

$$K_{\text{eff}} := \frac{K}{1 + (K-1)\rho}.$$

Proof. Let $Z = (Z_1, \dots, Z_K)^\top$ and let $\mathbf{1} \in \mathbb{R}^K$ be the all-one vector. The observation model can be written as

$$Z = \mathbf{1}Y + \varepsilon,$$

where

$$\varepsilon = U\mathbf{1} + (V_1, \dots, V_K)^\top$$

is Gaussian noise with covariance

$$\Sigma = \sigma_u^2 I_K + \sigma_c^2 \mathbf{1}\mathbf{1}^\top.$$

For a scalar Gaussian signal observed through a linear Gaussian channel,

$$I(Y; Z) = \frac{1}{2} \log \left(1 + \mathbf{1}^\top \Sigma^{-1} \mathbf{1} \right).$$

By the Sherman–Morrison formula,

$$\Sigma^{-1} = \frac{1}{\sigma_u^2} I_K - \frac{\sigma_c^2}{\sigma_u^2(\sigma_u^2 + K\sigma_c^2)} \mathbf{1}\mathbf{1}^\top.$$

Therefore,

$$\mathbf{1}^\top \Sigma^{-1} \mathbf{1} = \frac{K}{\sigma_u^2} - \frac{\sigma_c^2 K^2}{\sigma_u^2(\sigma_u^2 + K\sigma_c^2)} = \frac{K}{\sigma_u^2 + K\sigma_c^2}.$$

Thus,

$$I(Y; Z_{1:K}) = \frac{1}{2} \log \left(1 + \frac{K}{\sigma_u^2 + K\sigma_c^2} \right).$$

Using

$$\sigma^2 = \sigma_u^2 + \sigma_c^2, \quad \rho = \frac{\sigma_c^2}{\sigma^2},$$

we have

$$\sigma_u^2 + K\sigma_c^2 = \sigma^2(1 + (K-1)\rho).$$

Hence

$$\frac{K}{\sigma_u^2 + K\sigma_c^2} = \frac{1}{\sigma^2} \cdot \frac{K}{1 + (K-1)\rho}.$$

Defining

$$K_{\text{eff}} := \frac{K}{1 + (K-1)\rho}$$

gives the claimed expression.

Proposition 8 shows that nominal parallel width K can be much larger than the effective information width K_{eff} . When $\rho = 0$, the path-level noises are independent and

$$K_{\text{eff}} = K.$$

In this case, all paths contribute complementary observations. When $\rho \rightarrow 1$,

$$K_{\text{eff}} \rightarrow 1,$$

meaning that even many parallel paths behave like a single effective information source. More generally, for fixed $\rho > 0$,

$$\lim_{K \rightarrow \infty} K_{\text{eff}} = \frac{1}{\rho},$$

so increasing the number of paths has diminishing information returns when cross-path dependence is high.

This closed-form model illustrates why independent parallel sampling can be inefficient despite using many rollouts. If different paths produce highly correlated residual search information, increasing K does not linearly increase global decision information. In this model, collaboration is useful when it reduces the correlation of residual information that remains to be discovered after conditioning on the shared state. This does not require raw generations to become more diverse; rather, already shared conclusions are removed from the residual search problem, so subsequent paths can spend less effort rediscovering the same information.

I Prompts

We list the prompt templates used by our CPT implementation. The worker prompt is used for each reasoning branch, while the information-extraction prompt is used by the same policy model to distill reusable intermediate notes from partial branch transcripts. We distinguish the dataset-specific templates for Math, GPQA-Diamond, and LiveCodeBench below. Double braces in the templates denote fields that are populated at runtime.

I.1 Math Prompts

Case Analysis: Broadcast Notes vs. No-Broadcast Self-Discovery

Question. Let $\zeta = e^{2\pi i/13}$. Suppose $a > b > c > d$ are positive integers and

$$|\zeta^a + \zeta^b + \zeta^c + \zeta^d| = \sqrt{3}.$$

Compute the smallest possible value of $1000a + 100b + 10c + d$.

Answer. 7521, attained by $(a, b, c, d) = (7, 5, 2, 1)$. Pool broadcasting begins at Round 9 for the CPT branch.

Broadcast information.

- #47: the useful target is that the six pairwise positive differences are exactly $\{1, 2, 3, 4, 5, 6\}$, which gives the required cosine sum.
- #53: the configuration $\{z, z+1, z+4, z+6\}$ with $z = 1$ gives the minimal concrete set $\{1, 2, 5, 7\}$, i.e., $(a, b, c, d) = (7, 5, 2, 1)$.

Reasoning snapshots: received broadcast information vs. self-discovered information.

Before receiving the broadcast, the branch has only reduced the problem to a cosine-sum condition:

$$|S|^2 = 3 \Rightarrow 4 + 2D = 3 \Rightarrow D = -1/2,$$

where D is the sum of the six pairwise cosine terms. It first considers the real-sum/conjugate-pair simplification

$$T = \{m, 13 - m, n, 13 - n\},$$

but has not yet identified the difference-set certificate above.

AFTER POOL BROADCAST

CPT branch, after broadcast

Broadcast-guided snapshot. After receiving notes #47 and #53, the branch immediately tests the concrete set $\{1, 2, 5, 7\}$:

$$\begin{aligned} 7 - 5 = 2, \quad 5 - 2 = 3, \quad 2 - 1 = 1, \\ 7 - 2 = 5, \quad 7 - 1 = 6, \quad 5 - 1 = 4. \end{aligned}$$

Hence the six cosine terms are exactly the known complete set:

$$D = \sum_{k=1}^6 \cos(2\pi k/13) = -1/2.$$

This gives $|S|^2 = 4 + 2D = 3$ without further broad subset exploration.

After using the broadcast information, the branch only needs the final verification:

$$(a, b, c, d) = (7, 5, 2, 1), \quad 1000a + 100b + 10c + d = 7521.$$

NO-BROADCAST SELF-DISCOVERY

same prefix, no pool entries

... private enumeration over cosine/difference candidates ...

Rediscovery snapshot. Much later in the continuation, the no-broadcast trace writes: "Set $\{1, 2, 5, 7\}$: $a = 7, b = 5, c = 2, d = 1$. Differences: $7 - 5 = 2, 5 - 2 = 3, 2 - 1 = 1, 7 - 2 = 5, 7 - 1 = 6, 5 - 1 = 4$. Sum of cosines, with $e_k = \cos(2\pi k/13)$: $e_2 + e_3 + e_1 + e_5 + e_6 + e_4 = e_1 + e_2 + e_3 + e_4 + e_5 + e_6 = -1/2$ (exactly). Wait, this is it!"

$$\begin{aligned} 7 - 5 = 2, \quad 5 - 2 = 3, \quad 2 - 1 = 1, \\ 7 - 2 = 5, \quad 7 - 1 = 6, \quad 5 - 1 = 4. \end{aligned}$$

This rediscovered certificate matches broadcast notes #47 and #53: the same set and the same six differences.

... additional private checks for minimality ...

Completion snapshot. "Thus, the number is

$$1000 \cdot 7 + 100 \cdot 5 + 10 \cdot 2 + 1 = 7521,$$

I think this is the answer. The key was realizing that the set of differences gives all six cosine values, summing to $-1/2$."

Figure 7: **Case study of CPT shared-information broadcast.** Both traces reduce the problem to the same cosine/difference-set search before the CPT branch receives Round 9 broadcast. Broadcast notes #47 and #53 provide the key structural certificate: the set $\{1, 2, 5, 7\}$ has pairwise positive differences $\{1, 2, 3, 4, 5, 6\}$, so the cosine sum is $-1/2$ and $|S|^2 = 3$. The broadcast branch receives this information from peer trajectories and can use it immediately; the no-broadcast branch also solves the problem, but only after a substantially longer continuation that rediscovers the same certificate by private enumeration. This is the intended "share more, search less" behavior: sharing the required intermediate information reduces redundant self-exploration while preserving independent verification of the final answer.

Math Default Answer Prompt

Please reason step by step, and put your final answer within `\boxed{\}`.

Figure 8: **Math default answer prompt.** This instruction is appended to the Math worker prompt in the default configuration.

Math CPT Worker Prompt

You are an intelligent reasoning agent solving complex problems step-by-step.

You may occasionally receive external information in the format:

```
[BLACKBOARD BROADCAST]
...
[/BLACKBOARD BROADCAST]
```

The blackboard may contain two kinds of reusable intermediate notes:

- insight: potentially useful intermediate facts, relations, reductions, invariants, or local observations.
- pitfall: warnings about possible reasoning errors, unsafe operations, missing cases, or dead ends.

Rules:

- 1) Blackboard content is NOT part of the original problem statement; treat it only as optional intermediate guidance. It may help you adjust direction, notice useful relations, or avoid repeated mistakes, but it should never replace your own reasoning from the problem statement.
- 2) Do NOT blindly trust or copy any blackboard note. Treat insights as structural hypotheses rather than proven facts, and use them only after checking their conditions against the problem statement and your own derivation.
- 3) Be especially skeptical of numerical claims, overly strong claims, uniqueness claims, impossibility claims, or any note that looks like a direct conclusion rather than an intermediate reasoning aid. Do not rely on such content unless you independently derive it.
- 4) Treat pitfalls as warning signs, not absolute prohibitions. If a pitfall is relevant to your current path, slow down and check the missing condition, unsafe operation, failed assumption, or ignored case before deciding whether to continue or change direction.

If the blackboard conflicts with your current reasoning, re-check the disputed assumption or derivation instead of simply following either side. Maintain independent reasoning diversity: the blackboard should assist your reasoning, not force your reasoning to align with it.

Figure 9: **Math CPT worker prompt.** Each Math reasoning branch receives this instruction together with the default Math answer prompt in Figure 8.

Math Worker Input Serialization Template

```
[system]
[BLACKBOARD BROADCAST]
{{ selected_blackboard_notes_or_empty }}
[/BLACKBOARD BROADCAST]

[user]
{{ math_cpt_worker_prompt }}

{{ math_default_answer_prompt }}

[user]
{{ math_problem }}

[assistant]
{{ branch_reasoning_continuation }}
```

Figure 10: **Math worker input serialization template.** The shared-information block occupies the leading system message so that shared notes always appear before the Math branch continues its private reasoning trace.

Math Information-Extraction System Prompt

You are a strict Strategic Reasoning Distiller for a shared blackboard.

Your task is to extract only NEW, high-confidence, reusable intermediate notes from a partial solution transcript. These notes will be broadcast to other reasoning paths, so noisy, weak, or overly broad notes can actively harm future reasoning.

You may also receive:

[HISTORY BROADCAST]

...

[/HISTORY BROADCAST]

These are notes already extracted in previous rounds.

Core principles:

- Treat the transcript as a partial and possibly incomplete reasoning process. Focus only on intermediate information that remains locally valid and reusable.
- Before writing any note, verify that it is directly supported by the transcript, standard mathematical facts, or a clearly valid local derivation. All necessary conditions, assumptions, cases, and scopes must be stated explicitly. If another reasoning path could not safely reuse the note without reading the original transcript, omit it.
- Be conservative. Do not write guesses, intuitions, pattern-based claims, conclusions supported only by small examples, or statements stronger than what has actually been justified. If a claim is only partially supported, either narrow its scope precisely or omit it.
- Do not repeat or lightly paraphrase ideas already covered in HISTORY BROADCAST. Prefer fewer high-confidence notes over many medium-confidence notes; an empty block is better than a noisy block.

What to write:

- insight = a locally verified, reusable intermediate fact, relation, invariant, formula, reduction, or scoped observation
- pitfall = a concrete negative constraint or warning, such as a non-equivalent transformation, unsafe division, missing case, circular argument, unjustified generalization, dead-end strategy, or contradiction.
- For final-answer-like claims in the transcript, extract only the reusable intermediate relation, condition, or risk behind them, not the final result, total count, uniqueness claim, or impossibility claim itself.

Output format:

- Output ONLY:

[BB_WRITE]

...

[/BB_WRITE]

- Each bullet must start with:

- (type=insight|pitfall)

- One sentence per bullet.

- Keep the total output concise

- If no note passes the rules, output:

[BB_WRITE]

[/BB_WRITE]

Figure 11: **Math information-extraction system prompt.** This prompt instructs the model to extract concise, reusable, and conservative mathematical notes from a partial branch transcript.

I.2 GPQA-Diamond Prompts

GPQA-Diamond Default Answer Prompt

You are solving a difficult scientific multiple-choice question.
Use careful domain reasoning, check the assumptions in the problem statement, and compare all four answer choices before committing.
At the end, provide the final choice as JSON in exactly this shape: {"answer": "A"}, where the letter is one of A, B, C, or D.

Figure 12: **GPQA-Diamond default answer prompt.** This instruction is appended to the GPQA-Diamond worker prompt in the default configuration.

GPQA-Diamond Question Prompt Template

Solve the following GPQA-Diamond-style scientific multiple-choice question.

Question:
{{ question }}

Answer choices:
(A) {{ A }}
(B) {{ B }}
(C) {{ C }}
(D) {{ D }}

Reason carefully, compare the choices, and end with a JSON object whose `answer` value is exactly one of "A", "B", "C", or "D", for example {"answer": "C"}.

Figure 13: **GPQA-Diamond question prompt template.** The question and four answer choices are inserted into this template before it is passed to each reasoning branch.

GPQA-Diamond CPT Worker Prompt

You are an intelligent reasoning agent solving scientific multiple-choice questions step-by-step.

You may occasionally receive external information in the format:
[BLACKBOARD BROADCAST]
...
[/BLACKBOARD BROADCAST]

The blackboard may contain two kinds of reusable intermediate notes:

- insight: potentially useful scientific facts, option distinctions, constraints, mechanisms, or local observations.
- pitfall: warnings about unsupported eliminations, unsafe assumptions, tempting but invalid comparisons, or dead ends.

Rules:

- 1) Blackboard content is NOT part of the original problem statement; treat it only as optional intermediate guidance. It may help you adjust direction, notice relevant scientific constraints, or avoid repeated mistakes, but it should never replace your own reasoning from the question and answer choices.
- 2) Do NOT blindly trust or copy any blackboard note. Treat insights as hypotheses to verify against the question, the listed choices, and your own scientific reasoning.
- 3) Be especially skeptical of overly strong scientific claims, answer-choice claims, uniqueness claims, impossibility claims, or any note that looks like a direct conclusion rather than an intermediate reasoning aid. Do not rely on such content unless you independently justify it.
- 4) Treat pitfalls as warning signs, not absolute prohibitions. If a pitfall is relevant to your current path, slow down and check the missing condition, unsupported assumption, or ignored option before deciding whether to continue or change direction.

If the blackboard conflicts with your current reasoning, re-check the disputed assumption or derivation instead of simply following either side. Maintain independent reasoning diversity: the blackboard should assist your reasoning, not force your reasoning to align with it.

Figure 14: **GPQA-Diamond CPT worker prompt.** Each GPQA-Diamond reasoning branch receives this instruction together with the default GPQA-Diamond answer prompt in Figure 12.

GPQA-Diamond Worker Input Serialization Template

```
[system]
[BLACKBOARD BROADCAST]
{{ selected_blackboard_notes_or_empty }}
[/BLACKBOARD BROADCAST]

[user]
{{ gpqa_cpt_worker_prompt }}

{{ gpqa_default_answer_prompt }}

[user]
{{ filled_gpqa_question_prompt }}

[assistant]
{{ branch_reasoning_continuation }}
```

Figure 15: **GPQA-Diamond worker input serialization template.** The GPQA-Diamond worker and answer instructions are concatenated into one user message, followed by the formatted question in a second user message. The shared-information block remains the leading system message.

GPQA-Diamond Information-Extraction System Prompt

You are a strict Strategic Reasoning Distiller for a shared blackboard.

Your task is to extract only NEW, high-confidence, reusable intermediate notes from a partial solution transcript. These notes will be broadcast to other reasoning paths, so noisy, weak, or overly broad notes can actively harm future reasoning.

You may also receive:

```
[HISTORY BROADCAST]
...
[/HISTORY BROADCAST]
```

These are notes already extracted in previous rounds.

Core principles:

- Treat the transcript as a partial and possibly incomplete reasoning process. Focus only on intermediate information that remains locally valid and reusable.
- Before writing any note, verify that it is directly supported by the transcript, established scientific knowledge, or a clearly valid option comparison. All necessary conditions, assumptions, and scopes must be stated explicitly. If another reasoning path could not safely reuse the note without reading the original transcript, omit it.
- Be conservative. Do not write guesses, surface-level familiarity judgments, pattern-based claims, or statements stronger than what has actually been justified. If a claim is only partially supported, either narrow its scope precisely or omit it.
- Do not repeat or lightly paraphrase ideas already covered in HISTORY BROADCAST. Prefer fewer high-confidence notes over many medium-confidence notes; an empty block is better than a noisy block.

What to write:

- insight = a locally verified, reusable scientific fact, option distinction, constraint, mechanism, formula, or scoped observation.
- pitfall = a concrete negative constraint or warning, such as an unsupported elimination, ambiguous wording, missing condition, unjustified generalization, dead-end comparison, or contradiction.
- For answer-choice-like claims in the transcript, extract only the reusable relation, condition, or risk behind them, not the final choice itself.

Output format:

- Output ONLY:

```
[BB_WRITE]
...
[/BB_WRITE]
```

- Each bullet must start with:
 - (type=insight|pitfall)
- One sentence per bullet.
- Keep the total output concise
- If no note passes the rules, output:

```
[BB_WRITE]
[/BB_WRITE]
```

Figure 16: **GPQA-Diamond information-extraction system prompt.** This prompt instructs the policy model to extract concise, reusable, and conservative scientific notes from a partial GPQA-Diamond branch transcript.

I.3 LiveCodeBench Prompts

LiveCodeBench Default Answer Prompt

```
Reason about the programming problem step by step, then provide the final answer as a complete Python solution in exactly one ```python ... ``` code block. If starter code is provided, complete that function or class interface; otherwise write a full stdin/stdout program.
```

Figure 17: **LiveCodeBench default answer prompt.** This instruction is appended to the LiveCodeBench worker prompt in the default configuration.

LiveCodeBench Problem Prompt Template (with Starter Code)

```
### Question:
{{ question_content }}

### Format: Use the starter code below as the required scaffold for your solution, and return the completed Python code inside delimiters.
```python
{{ starter_code }}
```

### Answer: (use the provided format with backticks)
```

LiveCodeBench Problem Prompt Template (without Starter Code)

```
### Question:
{{ question_content }}

### Format: Write a complete Python program that reads from stdin and writes to stdout. Do not hard-code sample tests. Return the code inside delimiters.
```python
YOUR CODE HERE
```

### Answer: (use the provided format with backticks)
```

Figure 18: **LiveCodeBench problem prompt templates.** The implementation selects the upper template when starter code is provided and the lower template otherwise.

LiveCodeBench CPT Worker Prompt

You are an intelligent programming agent solving competitive-programming style code generation problems step-by-step.

You may occasionally receive external information in the format:

```
[BLACKBOARD BROADCAST]
...
[/BLACKBOARD BROADCAST]
```

The blackboard may contain two kinds of reusable intermediate notes:

- insight: potentially useful facts about the input/output contract, constraints, algorithmic structure, invariants, data structures, or implementation details.
- pitfall: warnings about possible bugs, invalid complexity assumptions, edge cases, unsafe parsing, or dead-end approaches.

Rules:

- 1) Blackboard content is NOT part of the original problem statement; treat it only as optional intermediate guidance. It may help you adjust the algorithm, check edge cases, or avoid implementation mistakes, but it should never replace your own reading of the problem statement and starter code.
- 2) Do NOT blindly trust or copy any blackboard note. Treat insights as hypotheses about the task, and use them only after checking their conditions against the problem statement, constraints, and examples.
- 3) Be especially skeptical of complexity claims, overly broad algorithm choices, final-code claims, or notes that ignore input bounds or hidden tests. Do not rely on such content unless you independently verify it.
- 4) Treat pitfalls as warning signs, not absolute prohibitions. If a pitfall is relevant to your current path, check the missing condition, unsafe operation, failed assumption, or ignored case before deciding whether to continue or change direction.

If the blackboard conflicts with your current reasoning, re-check the disputed assumption or implementation detail instead of simply following either side. Maintain independent reasoning diversity: the blackboard should assist your solution process, not force all paths into the same code.

Figure 19: **LiveCodeBench CPT worker prompt.** Each LiveCodeBench reasoning branch receives this instruction together with the default answer prompt in Figure 17.

LiveCodeBench Worker Input Serialization Template

```
[system]
[BLACKBOARD BROADCAST]
{{ selected_blackboard_notes_or_empty }}
[/BLACKBOARD BROADCAST]

[user]
{{ livecodebench_cpt_worker_prompt }}

{{ livecodebench_default_answer_prompt }}

[user]
{{ filled_livecodebench_problem_prompt }}

[assistant]
{{ branch_reasoning_continuation }}
```

Figure 20: **LiveCodeBench worker input serialization template.** The LiveCodeBench worker and answer instructions are concatenated into one user message, followed by the formatted programming problem in a second user message. The shared-information block remains the leading system message.

LiveCodeBench Information-Extraction System Prompt

You are a strict Programming Note Distiller for a shared blackboard.

Your task is to extract only NEW, high-confidence, reusable programming notes from a partial solution transcript. These notes will be broadcast to other solution attempts, so noisy, weak, or overly broad notes can actively harm future code.

You may also receive:

[HISTORY BROADCAST]

...

[/HISTORY BROADCAST]

These are notes already extracted in previous rounds.

Core principles:

- Treat the transcript as a partial and possibly incomplete programming attempt. Focus only on intermediate information that remains locally valid and reusable.
- Before writing any note, verify that it is directly supported by the transcript, the problem statement, the starter code, or a clearly valid local derivation. All necessary conditions, assumptions, cases, and scopes must be stated explicitly. If another solution path could not safely reuse the note without reading the original transcript, omit it.
- Be conservative. Do not write guesses, intuitions, claims supported only by small examples, or statements stronger than what has actually been justified. If a claim is only partially supported, either narrow its scope precisely or omit it.
- Do not repeat or lightly paraphrase ideas already covered in HISTORY BROADCAST. Prefer fewer high-confidence notes over many medium-confidence notes; an empty block is better than a noisy block.

What to write:

- insight = a locally verified, reusable programming fact, input/output rule, invariant, recurrence, reduction, complexity observation, data-structure choice, or implementation detail.
- pitfall = a concrete warning, such as an invalid parsing assumption, wrong boundary condition, unsafe mutation, missing case, excessive complexity, sample-only shortcut, or contradiction.
- For final-code-like claims in the transcript, extract only the reusable condition, invariant, interface requirement, or implementation risk behind them, not the final solution code or a "passes all tests" claim.

Output format:

- Output ONLY:

[BB_WRITE]

...

[/BB_WRITE]

- Each bullet must start with:
 - (type=insight|pitfall)
- One sentence per bullet.
- Keep the total output concise
- If no note passes the rules, output:

[BB_WRITE]

[/BB_WRITE]

Figure 21: **LiveCodeBench information-extraction system prompt.** This prompt instructs the policy model to extract concise, reusable, and conservative programming notes from a partial LiveCodeBench branch transcript.