ParaMind: End-to-End Parallel Reasoning for Large Language Models

Anonymous ACL submission

Abstract

Test-time scaling has emerged as a powerful way to improve the reasoning capability of large language models, typically by generating longer sequential chains of thought at inference time. While effective, this approach often reaches a bottleneck as test-time computation increases, resulting in a "test-time scaling plateau" where further computation provides only marginal gains. We argue that this ceiling often reflects a limitation of the scaling strategy rather than the model capability itself, a phenomenon we term *Tunnel Vision*, where imperfect early steps can lock the model into a suboptimal reasoning path. Motivated by this observation, we propose *ParaMind*, an end-to-end parallel reasoning framework that trains LLMs to generate multiple reasoning paths in parallel and aggregate them into a final answer within a single architecture. By exploring different lines of thought simultaneously, *ParaMind* effectively sidesteps the *Tunnel Vision* issue and unlocks the model's reasoning potential. On challenging reasoning benchmarks, ParaMind achieves substantial accuracy improvements (e.g. 6.5%-20.7% on AIME-24 with 1.5B model) over sequential-reasoning LLMs under the same budget of decoding tokens, without inducing additional computational cost. The reasoning latency can even be reduced by 38.7%-66.8% via batch decoding in on-device single-request settings.

1 Introduction

The remarkable progress of Large Language Models (LLMs) has been largely driven by the principle of scaling. This evolution began with pretraining compute scaling and has recently shifted to post-training or test-time compute scaling. Notable examples of test-time scaling, such as OpenAI o1 (OpenAI, 2024) and DeepSeek-R1 (DeepSeek-AI, 2025), have demonstrated that training the models to "think longer" (*i.e.* decode more tokens before generating the final answer) can unlock superior reasoning abilities for complex problems (Yang et al., 2025a; Team et al., 2025; Snell et al., 2024; Wu et al., 2025).

However, extending test-time compute does not lead to constant performance improvement in today's reasoning LLMs, where accuracy improvements diminish and eventually stagnate after a certain number of decoding steps. This has fueled discussions around "LLM overthinking" (Ghosal et al., 2025; Chen et al., 2025b; Fan et al., 2025; Cuadron et al., 2025; Li et al., 2025), where models expend excessive computation on problems, with the additional reasoning steps yielding minimal or no benefit to the final answer.

In this paper, we investigate the problem of test-time scaling bottleneck by raising the question: *Does the test-time scaling bottleneck stem from the inherent limitations of the model's capability, or from the test-time compute strategy?* Our empirical findings suggest that, under a fixed total decoding token budget, single-path sequential reasoning (as used in models like o1 and R1) can eventually be surpassed by simple parallel strategies (*e.g.* majority voting) as the token budget increases. This suggests that the test-time scaling bottleneck may arise not only from model capability, but also from how test-time compute is allocated.

Through a closer analysis of the reasoning process, we find that the reasoning performance is often constrained by the model's initial thoughts, a phenomenon we refer to as *Tunnel Vision*. Specifically, the first few tokens generated in a Chain-of-Thought (CoT) can lock the model into a suboptimal reasoning path, preventing it from discovering more effective ideas in subsequent decoding steps.

Based on these observations, we explore an alternative: executing the reasoning process in a parallel, multi-path manner. By generating independent thinking paths, this strategy can mitigate Tunnel Vision and encourage diversity of thought. Additionally, parallel thinking offers practical deployment

advantages, as the decoding process can be batched to better utilize memory bandwidth, which in turn leads to improved arithmetic intensity (Williams et al., 2009) (the ratio of floating-point operations to total data movement).

To put parallel thinking into practice, we introduce an end-to-end solution, ParaMind, which enables native parallel thinking in LLMs by allowing the model to generate diverse thoughts and aggregate them into a final answer. The major challenges to develop ParaMind include how to induce thought diversity and how to avoid thought conflict, which we address by introducing trainable control tokens to trigger distinct reasoning trajectories, thought-specific positional embeddings to distinguish different paths, and a two-phase attention mask design that enforces independence during reasoning and controlled integration during summarization. We evaluate ParaMind on math and coding benchmarks: AIME 2024, AIME 2025, AMC 2023, MATH-500 (Hendrycks et al., 2021), and LiveCodeBench v6 (Jain et al., 2024) against baselines including standard sequential reasoning (DeepSeek-AI, 2025), majority voting (Chen et al., 2024a), DeepConf (Fu et al., 2025), and re-prefilling. Our experiments show consistent accuracy improvements across benchmarks, with the parallel decoding structure providing additional latency benefits. ParaMind achieves higher accuracy than sequential LLMs by 12.3% and 7.5% for 1.5B and 7B models, respectively. Compared with majority voting, ParaMind further improves accuracy by 4.3% and 2.0%.

In summary, the contributions of our work are: (1) We empirically characterize the test-time scaling bottleneck in LLM reasoning and provide evidence that a narrow reasoning pathway, termed Tunnel Vision, is a contributing factor. (2) We propose ParaMind, an end-to-end framework for native parallel thinking that integrates trainable control tokens, thought-specific positional embeddings, and a learned summarization mechanism. (3) We demonstrate that ParaMind consistently improves accuracy over sequential reasoning and majority voting baselines on math and coding benchmarks.

2 Understanding the Scaling Bottleneck

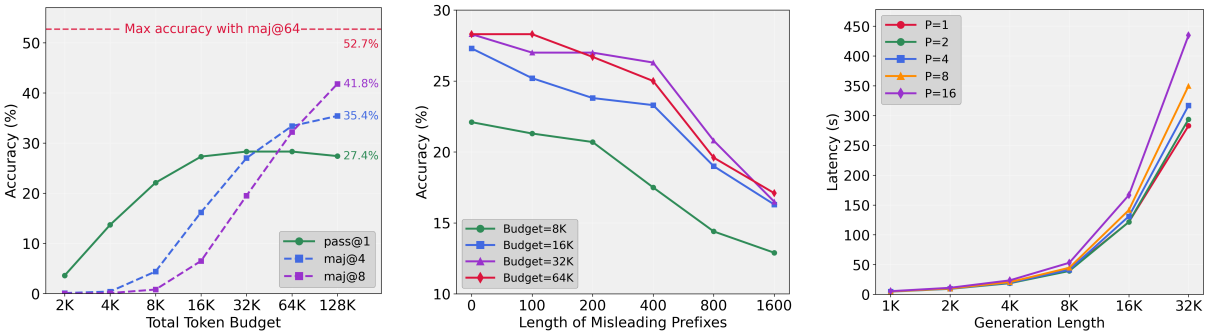
2.1 Is the Bottleneck Due to LLM Capability or Scaling Strategy?

To empirically ground our approach, we first characterize the limitations of conventional test-time scaling. We start by evaluating the DeepSeek-R1-distill-Qwen-1.5B model (DeepSeek-AI, 2025) on the AIME 2024 benchmark under various computational budgets. We control the budget by imposing a per-response token limit B on the reasoning path. If the model fails to terminate naturally, we truncate the output and force termination by appending a terminal token (`</think>`). We also evaluate majority voting (Chen et al., 2024a; Fu et al., 2025) over $P \in \{4, 8, 16, 32, 64\}$ parallel samples, with each sample allocated B/P tokens. For clarity, we plot only the results of $P = 4$ and $P = 8$ in the accuracy–budget curves, and report the maximum accuracy obtained with $P = 64$. The results, shown in Figure 1a, demonstrate that the performance of a single reasoning path (green line) quickly reaches the bottleneck, with additional tokens yielding negligible gains.

While some recent works attribute this phenomenon to "LLM overthinking" and attempt to solve it by compressing the model's output for more concise reasoning, these compressed models still encounter a bottleneck (Sun et al., 2025; Chen et al., 2025b; Fan et al., 2025; Cuadron et al., 2025). Different from these approaches, we investigate a potential underlying cause of this bottleneck to inform better test-time scaling strategies. And results in Figure 1a shows that majority voting can break through this bottleneck under the same total token budget, and the majority@64 with 2,048K total token budget (32K for each reasoning path), achieves a final accuracy far higher than the single-path approach. This gap suggests that *the bottleneck may not solely reflect a hard limit of the model's reasoning capacity, but could partly result from the single-path test-time scaling strategy.* In these experiments, allocating more test-time compute to a single-sequence LLM appears less effective than distributing it across multiple reasoning paths.

2.2 The Tunnel Vision of Sequential Test-Time Scaling

We hypothesize the bottleneck arises because an LLM's early token choices irreversibly commit it to a specific line of thought, making it difficult to escape initial errors. We call this Tunnel Vision: flawed initial reasoning locks the model into a suboptimal trajectory from which it cannot recover. To test this, we investigate the model's *recovery capacity* from erroneous starting points: For each AIME 2024 problem, we use DeepSeek-R1-Distill-Qwen-1.5B (DeepSeek-AI, 2025) to generate multiple



(a) Scaling Bottleneck (b) Tunnel Vision (c) Parallel Decoding Efficiency

Figure 1: Diagnosing the limitations of sequential reasoning and the potential of parallelism. All experiments use DeepSeek-R1-Distill-Qwen-1.5B on the AIME 2024 benchmark. (a) Scaling Bottleneck: Accuracy against the total number of token budgets (*e.g.* maj@4: the total token budget is the sum across all parallel paths). (b) Tunnel Vision: Ability to recover from its own potential mistakes with different lengths of misleading prefixes. (c) Parallel Decoding Efficiency: Latency taken to decode $P \in \{1, ..., 16\}$ parallel paths, each of length $n \in \{1\text{K}, \ldots, 32\text{K}\}$.

samples. From the samples that produce incorrect answers, we extract prefixes of its flawed reasoning at lengths of 0, 100, 200, 400, 800, and 1600 tokens. We then prompt the model to continue generating from these erroneous prefixes and measure its final accuracy by sampling 16 times and calculating the average accuracy. The results, plotted in Figure 1b, show a clear negative correlation: the longer the erroneous prefix, the lower the final accuracy. This supports our hypothesis that *the scaling bottleneck is at least partly attributable to Tunnel Vision, where flawed initial tokens can lock the model into a suboptimal reasoning path.* The longer the flawed prefix, the harder it is for the model to pivot to a correct solution, even with ample remaining budget.

3 Why Native Thought Parallelism?

Given the limitations of sequential reasoning exposed by Tunnel Vision, we are motivated to explore parallel reasoning as a natural alternative. Diverse independent reasoning paths can potentially reduce the risk that the model becomes stuck in a single suboptimal trajectory. A commonly used practical instantiation of this idea is majority voting (Chen et al., 2024a), which aggregates many independent answers into a final decision.

However, majority voting is a heuristic—it applies the simple, fixed rule of choosing the most frequent answer without learning to evaluate the quality of the reasoning that produced it. This approach has two important drawbacks: **(1) Poor generalization:** Voting is only applicable to tasks with easily quantifiable answers (*e.g.* , multiple-choice or numeric values), failing on open-ended domains like complex agentic workflows, coding, or long-form proofs. **(2) Information loss:** Majority voting considers only the final answer, instead of the reasoning path, ignoring the valuable rationale, evidence, and intermediate steps within each reasoning path, making it impossible to synthesize insights from the full reasoning process to arrive at a better conclusion.

To address these limitations, we explore **native parallel reasoning** as an end-to-end alternative. In this approach, we train the LLM to generate multiple reasoning paths in parallel and then analyze all paths to produce an answer. Instead of relying on a fixed heuristic, the model learns to directly process and synthesize the full token trajectories of its parallel thoughts. This makes the aggregation process trainable and path-aware, preserving information from each reasoning chain that would otherwise be discarded by answer-only methods. We formalize *Why native parallel reasoning has the potential to outperform majority voting* theoretically in Appendix A.10.

Moreover, parallel decoding is highly efficient, as LLM inference is memory-bound (parameter and KV cache loading) rather than compute-bound (Sadhukhan et al., 2025). Batching P paths amortizes memory accesses, enhancing arithmetic intensity and GPU utilization. Experiment with DeepSeek-R1-Distill-Qwen-1.5B using vLLM (Kwon et al., 2023) on an A800 GPU (Figure 1c) confirms: small P incurs similar latency to a single path, while $P = 16$ is under $2\times$. This efficiency makes parallel exploration scalable for overcoming Tunnel Vision and boosting reasoning performance.

Previous works on parallel computation typically rely on external verifiers for search (Snell et al., 2024; Ghosal et al., 2025), which introduce a scalability bottleneck. Several concurrent works (Zhao et al., 2025b; Yang et al., 2025b; Zheng et al., 2025; Fu et al., 2025) have also investigated the parallel reasoning mechanism of LLMs, while they struggle to achieve substantial accuracy improvement due to positional ambiguity, computational inefficiency, and/or limited thought diversity. More related works are discussed in Appendix A.2.

4 Model Design

Preliminaries of conventional sequential reasoning. We denote an LLM by π_θ, where θ is the set of model parameters. Given an input prompt of l tokens $x = \{x_i\}_{i=1}^{l}$. The LLM then autoregressively generates an output sequence $y = (y_1, y_2, \ldots, y_L)$ with the conditional probability: $\pi_\theta(y|x) = \prod_{t=1}^{L} \pi_\theta(y_t|x, y_{<t})$. For tasks requiring multi-step reasoning, the output y can be decomposed into a reasoning path r followed by a final answer a: $y = (r, a)$. During decoding, each new token y_t requires attention over the full context $x, y_{<t}$, which involves computing Key (K) and Value (V) tensors. To avoid recomputation when generating each y_t, LLMs often use a KV-cache to store K/V tensors.

4.1 ParaMind Workflow

As shown in Figure 2, our approach extends the sequential reasoning LLM paradigm by first generating a set of P distinct reasoning paths $\{r^{(1)}, r^{(2)}, \ldots, r^{(P)}\}$ for a single input x in parallel. Each individual reasoning path $r^{(i)}$ is a sequence of tokens representing a unique line of thought, sampled from the distribution: $\pi_\theta(r^{(i)}|x) = \prod_{t=1}^{L_i} \pi_\theta(r_t^{(i)}|x, s^{(i)}, r_{<t}^{(i)})$. Here, $s^{(i)}$ is a special control token that helps initiate a distinct reasoning path, which will be detailed in Section 4.2.

After generating these parallel paths, the model synthesizes them to produce a final answer, a. This answer is conditioned on both the original prompt x and the complete context of all preceding reasoning paths. Let $\mathcal{R} = (r^{(1)}, r^{(2)}, \ldots, r^{(P)})$ be the concatenation of all generated reasoning paths. The final answer a is then sampled from the model as follows: $\pi_\theta(a|x) = \prod_{t=1}^{La} \pi_\theta(a_t|x, \mathcal{R}, a_{<t})$. Crucially, ParaMind leverages the KV-caches from the parallel reasoning stage, eliminating the need to re-prefill the context and thereby offering significant computational savings compared to other methods.

4.2 Special Tokens for Boosting Thought Diversity

ParaMind needs to ensure diverse reasoning paths to avoid the trap of relying on a single sampled sequence. To achieve this, we introduce a set of trainable special control tokens: `<think i>`, `</think i>`, `<summary>`, and `</summary>` for $i \in \{1, \ldots, P\}$ to control the parallelization and merging operations. The `<think i>` token (denoted as $s^{(i)}$ in our equations) is placed at the beginning of each reasoning path, which leads the model to generate a distinct trajectory. Thus, the distribution of each reasoning path $\pi_\theta(r^{(i)}|x) = \prod_{t=1}^{L_i} \pi_\theta(r_t^{(i)}|x, s^{(i)}, r_{<t}^{(i)})$ will be conditioned by $s^{(i)}$. The closing `</think i>` token marks the end of a specific path, and the generation of the final answer is then wrapped within `<summary>` and `</summary>` tokens. This structured use of control tokens provides a mechanism to guide the model's generation process towards diverse and parallel lines of thought.

4.3 Thought-Specific Positional Embedding

Merging multiple reasoning paths poses challenges due to positional ambiguity. LLMs distinguish tokens based on their content and positional encoding. When multiple reasoning paths are generated in parallel, tokens at the same relative position (*e.g.* the t-th token in each reasoning path $r^{(i)}$) share identical positional encodings. This may cause confusion during summarization, as the model cannot differentiate which reasoning stream a token originated from.

Flattened Encoding: A naive solution assigns unique absolute positions across all paths: $m = l_x + i \cdot l_{\max} + t$, where l_x is the input length, i indexes the reasoning path, and t indexes the token position within that path. While this resolves positional collisions, it results in large positional indices as P increases. Typical positional encoding mechanisms such as Rotary Position Embedding (RoPE) (Su et al., 2024) encodes relative positions via rotations, and large index differences $|m - n|$ cause attention scores to decay. As a result, tokens from earlier paths (*i.e.* lower i of $r^{(i)}$) contribute less when generating the final answer, introducing imbalance across paths.

Sequence-Aware Positional Embedding: To address positional ambiguity in multi-response gen-

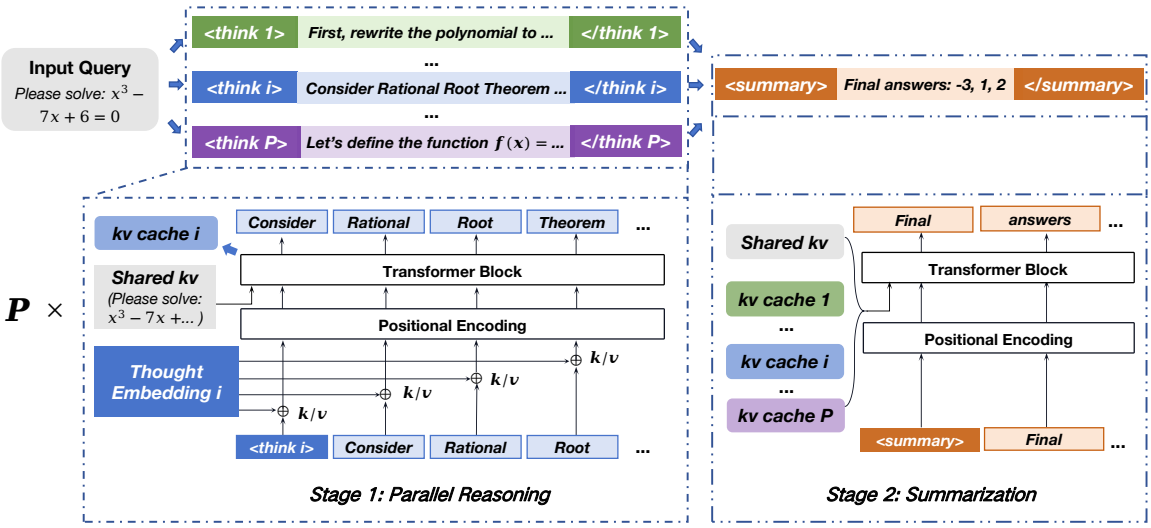


Figure 2: ParaMind architecture. For an input question, ParaMind processes it in two stages: (1) Parallel Reasoning: ParaMind generates P reasoning paths in parallel; (2) Summarization: ParaMind merges the reasoning paths by reusing their KV-caches to generate the final answer.

eration tasks, ParaMind separates different reasoning paths by augmenting the RoPE mechanism with learnable thought embeddings $\{T^{(j)}\}_{j=0}^{P}$. Specifically, we add the $T^{(j)}$ to the key and value embeddings of all tokens within the i-th reasoning path, which distinguishes each reasoning path at the summarizing phase. The thought embedding is added to the key before the RoPE rotation is applied. Let $\tilde{k}_t^{(j)}, \tilde{v}_t^{(j)}$ denote the cached key and value for token t at path j, respectively, from which the key and value vectors are formed as:

$$\tilde{k}_t^{(j)} = R_t(k_t^{(j)} + T^{(j)}) \qquad \tilde{v}_t^{(j)} = v_t^{(j)} + T^{(j)} \quad (1)$$

Here, l_{max} denotes the maximum token number for each reasoning path, and R_t is the corresponding RoPE rotation matrix. Using the RoPE property $(R_n)^T R_m = R_{m-n}$, the dot product attention score between a query q_n from the summary (at local position n) and a key $\tilde{k}_t^{(j)}$ from path j (at position m) is:

$$\begin{aligned} \text{score}(n, m) &= (R_n q_n^{(i)})^T \tilde{k}_m^{(j)} \\ &= (R_n q_n^{(i)})^T [R_m (k_m^{(j)} + T^{(j)})] \\ &= \underbrace{q_n^T R_{m-n} k_m^{(j)}}_{\text{Content-to-Content}} + \underbrace{q_n^T R_{m-n} T^{(j)}}_{\text{Content-to-Segment}} \end{aligned} \quad (2)$$

The Content-to-Content term is the standard RoPE attention score, which calculates the relevance between the query's content (q_n) and the key's content ($k_m^{(j)}$). This term is not related to the reasoning path number j and thus does not change when scaling parallel reasoning paths. Content-to-Segment term calculates the relevance between the query's content (q_n) and the learnable identity of the key's entire reasoning path ($T^{(j)}$). This allows the query to directly probe for the origin of the information. Because each reasoning path has a unique, trainable thought embedding, this term provides an unambiguous signal for the model to differentiate between parallel streams of text, solving the positional ambiguity.

5 Training and Deployment of ParaMind

5.1 Scalable Training Data Curation

ParaMind models are trained by fine-tuning existing sequential reasoning LLMs with synthetic parallel thought data. We design a scalable training data curation pipeline that consists of two key components: multi-path training data scaling and extensible special tokens training.

Multi-Path Training Data Scaling: We develop a simple yet effective high-quality parallel reasoning dataset by sampling multiple times from teacher reasoning LLM (*e.g.* DeepSeek-R1 (DeepSeek-AI, 2025)). For a query x and groundtruth answer a, let the $\hat{P}$ sampled answers denoted as $\{(r^{(1)}, a^{(1)}), (r^{(2)}, a^{(2)}), \ldots, (r^{(\hat{P})}, a^{(\hat{P})})\}$. We then concat each parallel answer into a groundtruth answer with the format of: $\hat{y}$ = (<think 1>$r^{(1)}$</think 1> , ..., <think $\hat{P}$>$r^{(\hat{P})}$</think $\hat{P}$> <summary>a</summary>). The resulting $Data_{r=1}^{\hat{P}} = (x, \hat{y})$ pairs are then used for SFT.

Extrapolative Special Tokens Training: Due to the high cost of teacher LLM inference, we are often faced with the situation where we cannot generate enough reasoning paths when creating $Data_{sft}$, *i.e.* , $\hat{P} < P$, where P denotes the maximum number of parallel reasoning paths supported at inference time. Thus, during SFT stage, LLMs have to learn to extrapolate to $(r^{(\hat{P}+1)}, \ldots, r^{(P)})$ with training data $Data_{r=1}^{\hat{P}}$. We develop a dynamic special token sampling method for extrapolative special tokens training. For each training batch, we randomly sample $\hat{P}$ special tokens from `<think i>`, $i \in \{1, \ldots, P\}$ and prepend these sampled tokens to the beginning of each reasoning sequence. Thus, each `<think_i>` specializes to induce diverse inference-time trajectories, despite training on only $\hat{P}$ paths.

5.2 Training and Inference Implementation

Attention Mask Design. ParaMind employs a two-phase attention mask design to enable parallel reasoning. In the reasoning phase, each path is decoded independently, with attention limited to the input prompt and its own generated tokens, thereby preventing inter-path interference. In the summarization phase, answer tokens attend to the full prompt, all reasoning paths, and prior answer tokens, allowing integration across paths while preserving autoregressive consistency. More details are shown in Appendix A.3.

The inference engine for ParaMind is built upon the vLLM framework (Kwon et al., 2023) to leverage its efficient PagedAttention mechanism for parallel decoding. The inference process is divided into two distinct phases:

Parallel Reasoning Phase: The engine processes the P reasoning paths concurrently as a single batch. This parallel decoding phase terminates for all paths as soon as the first parallel reasoning path is completed, namely any of the P paths generates an end-of-sequence (EOS) token. This termination strategy ensures all reasoning paths maintain an equal length, preventing processing imbalance. As empirically justified in Section A.8, this strategy yields the highest accuracy.

Summarization Phase: Following the parallel reasoning phase, the engine constructs an attention context spanning the KV caches of all P reasoning paths, eliminating the need for costly re-prefilling. Leveraging vLLM's PagedAttention, this step is performed with zero data copying, as the summary sequence can directly reference the memory blocks of all preceding paths.

6 Experiments

6.1 Experimental Setup

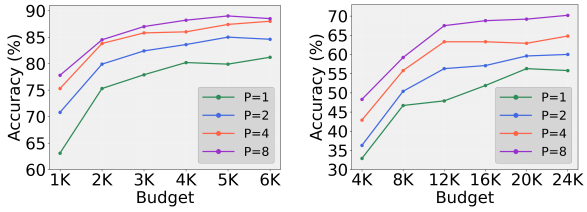
Training. Our math experiments use Qwen-2.5 (Qwen et al., 2025) 1.5B and 7B models distilled from DeepSeek-R1 (DeepSeek-AI, 2025) (denoted R1-1.5B and R1-7B). Coding experiments use DeepSeek-R1-Distill-Qwen-1.5B. For math, we construct 6.2K problem-solution pairs with $\hat{P} = 6$ reasoning paths each; during training, we randomly choose $P \in \{2, 4, 6\}$ paths per sample. For code, we use 50K instances from OpenCodeReasoning (Ahmad et al., 2025) with $\hat{P} \in \{2, 4\}$ paths. Details are in Appendix A.4.

Baselines. *(1) Sequential:* Direct reasoning with original models. *(2) Majority Voting:* P independent paths with majority answer (Chen et al., 2024a). *(3) DeepConf-high/low:* Confidence-aware baselines for parallel reasoning that leverage model-internal confidence signals to prune low-quality traces. Traces are aggregated using weighted voting, where each vote is weighted by the path's confidence score. High/low differs in retention ratio η, keeping top 10% (low) or top 90% (high) highest-confidence traces for the final vote. *(4) Re-Prefilling:* Generate P paths, concatenate them, and feed the full context into the model for summarization. This mimics ParaMind 's summarization but is inefficient since KV caches are not reused (Section 6.2). We do not include majority-voting for coding experiments since program correctness is often non-trivial to vote on.

Benchmarks. AIME 2024, AIME 2025, AMC 2023, MATH-500 (Hendrycks et al., 2021), and LiveCodeBench v6 (Jain et al., 2024). Each reasoning path is limited to B tokens. If a model reaches this budget without naturally stopping, we enforce termination and then initiate the summarization stage by adding the (`<summary>`) token. Details are in Appendix A.5.

6.2 Scaling Performance

Table 1 and Table 2 compare ParaMind with baseline methods for math and coding tasks. Compared with sequential LLMs, ParaMind improves accuracy by up to 14.5% (1.5B) and 8.3% (7B) on AIME 2024, and by 3.6% (1.5B) and 8.8% (7B) on AIME 2025 at each token length on average (*e.g.* 2×16K vs. 32K), demonstrating the effective-



(a) ParaMind-1.5B Math-500. (b) ParaMind-7B AIME-24.

Figure 3: ParaMind scaling performances.

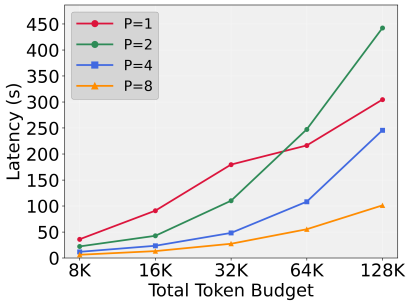


Figure 4: ParaMind Latency under different budgets.

ness of parallel reasoning. This indicates that the summarization stage captures a richer aggregation strategy than vote counting. For coding tasks, ParaMind outperforms original LLM and data ablation LLM (trained on the same dataset with ParaMind) with P increases, showing the potential to handle various tasks.

We analyze how performance scales with the number of parallel paths in Figure 3a and 3b, where increasing the path count consistently yields higher accuracy at larger generation budget. For sequential reasoning LLM ($P = 1$), expanding the token budget beyond 32K yields no further accuracy gains, whereas ParaMind continues to improve. These results suggest that ParaMind can continue to benefit from additional compute beyond the point where sequential reasoning models typically see diminishing returns.

We further analyze the relations between ParaMind and majority voting in detail. The result is shown in Table 3. We find that ParaMind does not conflict with the majority voting. Instead, it can be combined with majority voting to achieve higher accuracy. The highest accuracy of ParaMind-1.5B+maj@8 can reach 66.7% and 60.0% on AIME 2024 with $P = 4$ and $P = 8$, gaining 23.4% and 11.9% accuracy improvements against pass@1. (See Table 9 for results on 7B+maj@k)

Inference Efficiency. Figure 4 shows the latency of ParaMind under different total token budgets. Under a large budget (*e.g.* , 128K), the latency of high parallel size (*e.g.* , 8×16K) is much less than sequential scaling. This is because the decoding phase is typically bounded by memory bandwidth, and increasing the number of parallel reasoning paths does not increase data movement operations. Our experiments show that the reasoning latency on $128K$ can even be reduced by 66.8% with $8 * 16K$. These results suggest that parallel scaling can improve accuracy within acceptable inference latency.

6.3 Ablation Study

Train Data: We test how much the performance gain of ParaMind attribute to the training data. Specifically, we use the same data of ParaMind (6 samples for each question) to finetune the original LLM, without changing any other setting. Table 5 shows that finetuning does not improve performance, with results even slightly worse than the original LLM. ParaMind, on the other hand, outperforms the fine-tuned variant across all budgets, confirming its effectiveness. This ablation provides evidence that the performance gains stem primarily from the parallel-then-summarize architecture rather than from the distilled training data alone.

Summarization Mechanism: We compare our trainable summarization mechanism against alternative aggregation strategies like majority voting and a heuristic *Shortest-of-N*, which selects the final answer from the shortest reasoning path among the N generated paths (motivated by the first termination strategy in ParaMind). We also report the pass@P metric, which measures the fraction of problems solved correctly by at least one of the P generated reasoning paths. We apply these strategies to the same set of reasoning paths generated by ParaMind under a fixed budget of 16K tokens per path. As shown in Table 4, our end-to-end summarization outperforms all alternatives.

Thought Embedding and Special Tokens: We conduct ablation studies of thought embedding (by removing the thought embeddings) and special tokens (by replacing every `<think i>`, `</think i>` with `<think>`, `</think>`). We find that without special tokens and thought embeddings, the accuracy of ParaMind drops 0.6%-3.1%, 1.4%-4.3% respectively, demonstrating the contribution of these designs. Detailed analysis is in Appendix A.9.

7 Conclusion

Our work identifies a limitation of conventional sequential test-time compute scaling of LLMs named

Method	Base model: R1-1.5B				Base model: R1-7B				Avg.
	AIME 24	AIME 25	AMC 23	MATH	AIME 24	AIME 25	AMC 23	MATH	
Seq. (16K)	26.1	22.4	67.1	81.2	51.9	37.9	88.4	91.2	58.3
Seq. (32K)	28.3	24.5	68.9	81.8	55.5	37.9	89.8	92.0	59.8
Seq. (64K)	27.1	25.5	67.7	81.7	56.0	39.6	89.8	92.5	60.0
Seq. (128K)	27.4	22.1	68.0	81.8	52.7	40.4	89.8	92.6	59.4
Maj. (2×16K)	25.9	23.0	67.0	81.4	52.3	38.3	88.4	91.4	58.5
Maj. (4×16K)	32.9	27.5	74.3	86.7	60.6	43.1	92.2	93.5	63.9
Maj. (8×16K)	41.0	31.8	79.8	89.0	**68.8**	49.6	93.1	94.2	68.4
DeepConf. Low (2×16K)	28.0	23.5	67.9	82.0	53.1	40.6	88.4	92.8	59.5
DeepConf. Low (4×16K)	34.1	26.6	72.0	84.5	59.2	40.9	89.4	92.3	62.4
DeepConf. Low (8×16K)	37.7	28.5	74.7	86.0	62.7	42.7	90.9	90.6	64.2
DeepConf. High (2×16K)	28.0	23.5	67.9	82.0	53.1	40.6	88.4	92.8	59.5
DeepConf. High (4×16K)	35.9	**27.7**	75.7	87.3	63.1	44.4	**92.5**	93.5	65.0
DeepConf. High (8×16K)	44.1	**33.3**	79.5	**89.8**	68.4	49.8	**93.3**	93.8	69.0
Rep. (2×16K)	30.4	**26.7**	70.6	60.8	42.9	33.8	88.1	63.8	52.1
Rep. (4×16K)	24.2	25.8	61.3	58.6	43.3	33.3	86.3	63.2	49.5
Rep. (8×16K)	14.2	13.3	60.0	55.3	43.3	31.7	91.9	63.7	46.7
Para. (2×16K)	**34.8**	24.2	**73.1**	**87.5**	**57.1**	**46.0**	**89.5**	**93.2**	**63.2**
Para. (4×16K)	**43.3**	26.7	**80.8**	**88.7**	**63.3**	**46.9**	91.7	**94.2**	**67.0**
Para. (8×16K)	**48.1**	31.9	**83.1**	89.7	**68.8**	**51.3**	**93.3**	**94.5**	**70.1**

Table 1: Accuracy of ParaMind and baselines (Pass@1 %). **Avg.** is the mean across all datasets and models. For parallel methods, we bold the best and underline the second best.

Model	Original Sequential LLM				Fine-tuned Sequential LLM				**ParaMind-1.5B**		
Budget	12K	24K	48K	96K	12K	24K	48K	96K	2×12K	4×12K	8×12K
Accuracy	18.3	18.9	18.9	17.7	16.5	16.0	16.0	16.7	18.7	19.4	**20.1**

Table 2: Pass@1 accuracy of ParaMind, original sequential LLM, and the sequential LLM finetuned with the same data as ours on LiveCodeBench v6.

.

	P=1	P=2	P=4	P=8
pass@1	26.1	34.8	43.3	48.1
maj@4	32.9	42.5	53.0	56.3
maj@8	41.0	50.1	61.7	59.9
maj@16	**47.8**	**56.7**	**66.7**	**60.0**

Table 3: Performance of ParaMind-1.5B with majority voting on AIME 2024. P: number of parallel reasoning paths; $maj@k$: majority voting with k samples.

	Strategy				Pass Rate	
P	**Maj**	**SoN**	**M+S**	**Para.**	**pass@1**	**pass@P**
2	26.7	33.3	33.3	**34.8**	26.3	35.8
4	30.0	38.4	39.2	**43.3**	24.6	49.2
8	35.8	40.8	40.0	**48.1**	22.8	56.7

Table 4: Ablation study for aggregation strategies on AIME 2024. SoN: Shortest-of-N; $M + S$: SoN if no majority identified; $Para.$: ParaMind-1.5B

	A24	**A25**	**AMC**	**MATH**	**Avg.**
R1-1.5B-SFT (Same dataset with ParaMind)					
Seq. (16K)	26.3	18.5	66.0	81.1	48.0
Seq. (32K)	22.9	22.1	64.1	77.6	46.7
Seq. (64K)	25.8	17.3	62.2	77.6	45.7
Seq. (128K)	24.8	21.9	63.6	78.6	47.2
Maj. (2×16K)	26.0	18.1	66.3	81.0	47.9
Maj. (4×16K)	32.2	23.4	72.1	86.5	53.6
Maj. (8×16K)	42.5	27.1	79.8	89.2	59.7
Rep. (2×16K)	23.3	16.3	65.6	76.8	45.5
Rep. (4×16K)	15.0	11.7	55.6	70.6	38.2
Rep. (8×16K)	15.8	9.2	58.8	66.6	37.6
ParaMind-1.5B					
Para. (2×16K)	**34.8**	**24.2**	**73.1**	**87.5**	**54.9**
Para. (4×16K)	**43.3**	**26.7**	**80.8**	**88.7**	**59.9**
Para. (8×16K)	**48.1**	**31.9**	**83.1**	**89.7**	**63.2**

Table 5: Train data ablation result: Pass@1 accuracy (%) of the 1.5B sequential LLM fine-tuned with the same dataset as ParaMind.

"Tunnel Vision". We then introduce ParaMind, a framework for native parallel reasoning that sidesteps the limitation by generating and aggregating multiple thought paths simultaneously. On math and coding benchmarks, ParaMind consistently outperforms sequential reasoning and majority voting baselines under matched token budgets.

Limitations

Our study has several limitations. First, due to the computational resource constraints, our evaluation is currently limited to two model sizes (1.5B and 7B) from the same model family (Qwen-2.5 distilled from DeepSeek-R1). Second, the current training pipeline relies on synthetic multi-path supervision generated by a teacher model. This design makes training scalable, but it may also inherit biases or reasoning styles from the teacher, and its data efficiency relative to stronger supervision or reinforcement learning remains underexplored, which we plan to address in the future work. Third, our diversity analysis of the control tokens is preliminary; a more thorough investigation into whether and how the special tokens induce semantically distinct reasoning strategies would strengthen our understanding of the method. Fourth, our analysis of the mechanism behind performance gains is still incomplete. In particular, while the results suggest that control tokens and thought-specific embeddings improve diversity and aggregation, we do not yet provide a full characterization of whether the learned parallel paths correspond to genuinely distinct reasoning strategies, nor how the summarization module resolves conflicts across paths. We leave broader model coverage, more diverse task settings, deeper mechanistic analysis, and training with reinforcement learning or stronger feedback signals to future work.

References

Wasi Uddin Ahmad, Sean Narenthiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. 2025. Opencodereasoning: Advancing data distillation for competitive coding. *arXiv preprint arXiv:2504.01943*.

Marianne Arriola, Subham Sekhar Sahoo, Aaron Gokaslan, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Justin T Chiu, and Volodymyr Kuleshov. 2025. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*.

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

Lingjiao Chen, Jared Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. 2024a. Are more llm calls all you need? towards the scaling properties of compound ai systems. In *Advances in Neural Information Processing Systems*, volume 37, pages 45767–45790. Curran Associates, Inc.

Mouxiang Chen, Binyuan Hui, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Jianling Sun, Junyang Lin, and Zhongxin Liu. 2025a. Parallel scaling law for language models. *Preprint*, arXiv:2505.10475.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025b. Do not think that much for 2+3=? on the overthinking of o1-like llms. *Preprint*, arXiv:2412.21187.

Xinyun Chen, Ryan Andrew Chi, Xuezhi Wang, and Denny Zhou. 2024b. Premise order matters in reasoning with large language models. In *Forty-first International Conference on Machine Learning*.

Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, Nicholas Thumiger, Aditya Desai, Ion Stoica, Ana Klimovic, Graham Neubig, and Joseph E. Gonzalez. 2025. The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks. *Preprint*, arXiv:2502.08235.

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

Yifu Ding, Wentao Jiang, Shunyu Liu, Yongcheng Jing, Jinyang Guo, Yingjie Wang, Jing Zhang, Zengmao Wang, Ziwei Liu, Bo Du, Xianglong Liu, and Dacheng Tao. 2025. Dynamic parallel tree search for efficient LLM reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11233–11252, Vienna, Austria. Association for Computational Linguistics.

Chenrui Fan, Ming Li, Lichao Sun, and Tianyi Zhou. 2025. Missing premise exacerbates overthinking: Are reasoning models losing critical thinking skill? *Preprint*, arXiv:2504.06514.

Guhao Feng, Yihan Geng, Jian Guan, Wei Wu, Liwei Wang, and Di He. 2025. Theoretical benefit and limitation of diffusion language model. *Preprint*, arXiv:2502.09622.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. 2025. Deep think with confidence. *arXiv preprint arXiv:2508.15260*.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *CoRR*, abs/2502.05171.

Soumya Suvra Ghosal, Souradip Chakraborty, Avinash Reddy, Yifu Lu, Mengdi Wang, Dinesh Manocha, Furong Huang, Mohammad Ghavamzadeh, and Amrit Singh Bedi. 2025. Does thinking more always help? understanding test-time scaling in reasoning models. *Preprint*, arXiv:2506.04210.

Google. 2025. Gemini 2.5: Our most intelligent ai model. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *Preprint*, arXiv:2501.04519.

Qi Guo, Jianing Wang, Deyang Kong, Xiangyu Xi, Jianfei Zhang, Yi Lu, Jingang Wang, Wei Wang, Shikun Zhang, and Wei Ye. 2026. Ope: Overcoming information saturation in parallel thinking via outline-guided path exploration. *arXiv preprint arXiv:2602.08344*.

Nathan Habib, Clémentine Fourrier, Hynek Kydlíček, Thomas Wolf, and Lewis Tunstall. 2023. Lighteval: A lightweight framework for llm evaluation.

Zhengfu He, Tianxiang Sun, Qiong Tang, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. 2023. DiffusionBERT: Improving generative masked language models with diffusion models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4521–4534, Toronto, Canada. Association for Computational Linguistics.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, and 1 others. 2025. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*.

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

Hugging Face. 2025. Open r1: A fully open reproduction of deepseek-r1.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Sheng Jia, Xiao Wang, and Shiva Prasad Kasiviswanathan. 2025. Training large language models to reason in parallel with global forking tokens. *arXiv preprint arXiv:2510.05132*.

Tian Jin, Ellie Y. Cheng, Zack Ankner, Nikunj Saunshi, Blake M. Elias, Amir Yazdanbakhsh, Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. 2025. Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding. *Preprint*, arXiv:2502.11517.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Bespoke Labs. 2025. Bespoke-stratos: The unreasonable effectiveness of reasoning distillation. www.bespokelabs.ai/blog/bespoke-stratos-the-unreasonable-effectiveness-of-reasoning-distillation. Accessed: 2025-01-22.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Zheng Li, Qingxiu Dong, Jingyuan Ma, Di Zhang, and Zhifang Sui. 2025. Selfbudgeter: Adaptive token allocation for efficient llm reasoning. *arXiv preprint arXiv:2505.11274*.

Long Lian, Sida Wang, Felix Juefei-Xu, Tsu-Jui Fu, Xiuyu Li, Adam Yala, Trevor Darrell, Alane Suhr, Yuandong Tian, and Xi Victoria Lin. 2025. Threadweaver: Adaptive threading for efficient parallel reasoning in language models. *arXiv preprint arXiv:2512.07843*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

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

OpenAI. 2024. Openai o1 system card.

OpenAI. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. 2025. Learning adaptive parallel reasoning with language models. *Preprint*, arXiv:2504.15466.

Xiangyu Qi, Ashwinee Panda, Kaifeng Lyu, Xiao Ma, Subhrajit Roy, Ahmad Beirami, Prateek Mittal, and Peter Henderson. 2025. Safety alignment should be made more than just a few tokens deep. In *The Thirteenth International Conference on Learning Representations*.

Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. 2025. Optimizing test-time compute via meta reinforcement fine-tuning. *Preprint*, arXiv:2503.07572.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazarian, Anton Sinitsin, Denis Kuznedelev, and Dan Alistarh. 2025. Hogwild! inference: Parallel llm generation via concurrent attention. *Preprint*, arXiv:2504.06261.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian Yen, Avner May, Tianqi Chen, and Beidi Chen. 2025. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *International Conference on Representation Learning*, volume 2025, pages 6835–6850.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

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

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Yi Sun, Han Wang, Jiaqiang Li, Jiacheng Liu, Xiangyu Li, Hao Wen, Yizhen Yuan, Huiwen Zheng, Yan Liang, Yuanchun Li, and Yunxin Liu. 2025. An empirical study of llm reasoning ability under strict output length constraint. *arXiv preprint arXiv:2504.14350*.

Kimi Team and 1 others. 2025. Kimi k1.5: Scaling reinforcement learning with llms. *Preprint*, arXiv:2501.12599.

Guanghan Wang, Yair Schiff, Subham Sahoo, and Volodymyr Kuleshov. 2025. Remasking discrete diffusion models with inference-time scaling. *arXiv preprint arXiv:2503.00307*.

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*.

Hao Wen, Xinrui Wu, Yi Sun, Feifei Zhang, Liye Chen, Jie Wang, Yunxin Liu, Ya-Qin Zhang, and Yuanchun Li. 2025. Budgetthinker: Empowering budget-aware llm reasoning with control tokens. *arXiv preprint arXiv:2508.17196*.

Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, Austin, Texas. Association for Computational Linguistics.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2025. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *Preprint*, arXiv:2408.00724.

Haotian Xu, Xing Wu, Weinong Wang, Zhongzhi Li, Da Zheng, Boyuan Chen, Yi Hu, Shijia Kang, Jiaming Ji, Yingying Zhang, Zhijiang Guo, Yaodong Yang, Muhan Zhang, and Debing Zhang. 2025. Redstar: Does scaling long-cot data unlock better slow-reasoning systems? *Preprint*, arXiv:2501.11284.

Rongwu Xu, Zehan Qi, and Wei Xu. 2024. Preemptive answer "attacks" on chain-of-thought reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 14708–14726, Bangkok, Thailand. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, and 1 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, and 1 others. 2024. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. 2025b. Multiverse: Your language models secretly decide how to parallelize and merge generation. *arXiv preprint arXiv:2506.09991*.

Shinn Yao, Jeffrey Zhao, Shixiang Yu, and et al. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Jiasheng Ye, Zaixiang Zheng, Yu Bao, Lihua Qian, and Quanquan Gu. 2023. Diffusion language models can perform many tasks with scaling and instruction-finetuning. *arXiv preprint arXiv:2308.12219*.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *Preprint*, arXiv:2503.14476.

Yanfang Zhang, Yiliu Sun, Yibing Zhan, Dapeng Tao, Dacheng Tao, and Chen Gong. 2025. Large language models as an indirect reasoner: Contrapositive and contradiction for automated reasoning. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 5040–5057, Abu Dhabi, UAE. Association for Computational Linguistics.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. 2025a. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*.

Wenting Zhao, Pranjal Aggarwal, Swarnadeep Saha, Asli Celikyilmaz, Jason Weston, and Ilia Kulikov. 2025b. The majority is not always right: Rl training for solution aggregation. *arXiv preprint arXiv:2509.06870*.

Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, Xinyu Yang, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, and 1 others. 2025. Parallel-r1: Towards parallel thinking via reinforcement learning. *arXiv preprint arXiv:2509.07980*.

Jason Zhu and Hongyu Li. 2025. Towards concise and adaptive thinking in large reasoning models: A survey. *Preprint*, arXiv:2507.09662.

A Appendix

A.1 LLM Usage Statement

Large language models (*e.g.* , ChatGPT) were used only for minor language polishing and grammar correction. All ideas, experimental design, data analysis, and writing were conducted by the authors.

A.2 Related Works

A.2.1 Sequential Test-Time Scaling

Recent advances in test-time scaling seek to improve LLM reasoning by increasing computational depth during decoding, primarily through reinforcement learning (RL) (OpenAI, 2024; DeepSeek-AI, 2025; Google, 2025; Yang et al., 2025a; Team et al., 2025; Zhao et al., 2025a; Yu et al., 2025) and supervised fine-tuning (SFT) (Muennighoff et al., 2025; Ye et al., 2025). RL-based methods (OpenAI, 2024; Shao et al., 2024) encourage LLMs to allocate more computation to promising reasoning paths by encouraging self-reflection and iterative trial. Other approaches distill long-form rationales from larger teacher models into smaller student models, enabling deeper internal reasoning through fine-tuned CoT supervision (Labs, 2025; Ye et al., 2025; Xu et al., 2025; Geiping et al., 2025). While these methods significantly enhance LLM performance on complex tasks, they often suffer from increased inference latency and compute consumption due to long output sequences (Sun et al., 2025; Zhu and Li, 2025; Qu et al., 2025; Wen et al., 2025). Moreover, excessively long reasoning traces may introduce "overthinking" effects such as repetition or hallucination (Chen et al., 2025b; Ghosal et al., 2025). Besides, recent works have also shown that sequential reasoning LLMs are brittle to reasoning order (Chen et al., 2024b) or shallow token attacks (Xu et al., 2024; Qi et al., 2025). In contrast, our method introduces a new dimension of inference-time scaling—width—by executing multiple reasoning paths in parallel and summarizing them. This approach preserves reasoning efficiency while avoiding long single-path decoding.

A.2.2 Search-Based Methods for Parallel Reasoning

Parallel Decoding LLMs improve reasoning by sampling multiple tokens at each step to accelerate LLM inference and/or improving LLM performance. Early techniques include beam search (Wiseman and Rush, 2016), self-consistency (Wang et al., 2023), speculative decoding (Leviathan et al., 2023; Chen et al., 2023) and majority voting (Chen et al., 2024a). Recent advancements include Best-of-N (Lightman et al., 2023), Tree of Thoughts (ToT) (Yao et al., 2023; Ding et al., 2025), and Monte Carlo Tree Search (MCTS) (Snell et al., 2024; Guan et al., 2025). These approaches typically require an external verifier to evaluate and rank candidate completions, increasing computational cost and often relying on domain-specific or manually constructed reward signals. Our method departs from these paradigms by generating multiple reasoning trajectories internally and merging them using a lightweight summarization step, without requiring external verifiers or retraining.

A.2.3 Natively Parallel Generation Methods

Another line of work focuses on empowering LLMs to generate multiple tokens at each decoding iteration to accelerate LLMs theoretically. Diffusion-based language models (He et al., 2023; Ye et al., 2023; Zhao et al., 2025a; Wang et al.,

2025; Arriola et al., 2025) sample multiple tokens in parallel during each diffusion step. While these methods can theoretically enable parallel generation, recent theoretical analyses (Feng et al., 2025) shows that for tasks involving sequential dependencies (*e.g.* reasoning), the number of required diffusion steps can scale linearly with sequence length, undermining their efficiency. PARSCALE (Chen et al., 2025a) investigates architectural parallelism by duplicating the input multiple times, applying distinct transformations, and aggregating outputs token-wise. However, this approach still requires architectural changes and specialized continual pretraining. In contrast, our approach retains the standard LLM architecture and introduces parallelism at the reasoning level by generating and caching multiple distinct chains of thought, which are later summarized into a final answer. Several methods (Jia et al., 2025; Zhang et al., 2025; Guo et al., 2026) have been proposed to enhance the diversity of different paths during parallel sampling, thus reducing redundancy between different paths. They can be easily embedded into our framework for efficient and adaptive summarization. Other works (Yang et al., 2025b; Pan et al., 2025; Rodionov et al., 2025; Jin et al., 2025; Lian et al., 2025) propose to automatically identify subtasks that can be solved in parallel. While effective for compositional tasks, it relies on explicit subtask decomposition, and these works focus on efficiency rather than accuracy improvement. ParaMind, on the other hand, does not assume any subtask structure and improves both efficiency and accuracy by mitigating single-path failure cases (*e.g.* hallucinations or local optima) via diversity in reasoning. By integrating multiple KV caches in a summarization stage, our method scales inference without sacrificing correctness or requiring verifier models.

A.3 Attention Mask Design

To enable efficient parallel reasoning in existing LLM infrastructures during both training and inference, ParaMind adopts a two-phase attention mask design. During the reasoning phase, each reasoning path is decoded independently, with attention restricted to the input prompt and its own token history. Let $M_{i,j}$ denote the attention mask between the index i and index j, where attention score can be calculated as: $A_{i,j} = \text{Softmax}\left(\frac{q_i \cdot k_j + M_{i,j}}{\sqrt{d_k}}\right)$. The attention mask for the i-th reasoning path ($r^{(i)}$) is defined as:

$$M^{r^{(i)}}_{t,j} = \begin{cases} 0, & \text{if } j \leq t \text{ and } j \in \{1, \ldots, l_x\} \\ & \cup \text{Ind}_i \\ -\infty, & \text{otherwise} \end{cases} \tag{3}$$

where l_x is the length of the input prompt and Ind_i is the index range for tokens in the i-th reasoning path. This enforces independence across reasoning paths by blocking inter-path attention.

During the summarization phase, where each answer token attends to the entire prompt, all reasoning paths, and previously generated answer tokens. The summarization attention mask is defined as:

$$M^{A}_{t,j} = \begin{cases} 0, & \text{if } j \leq t \text{ and } j \in \{1, \ldots, l_x\} \cup \\ & \bigcup_{i=1}^{P} \text{Ind}_i \cup \text{Ind}_a \\ -\infty, & \text{otherwise} \end{cases} \tag{4}$$

where Ind_a denotes the index range of the answer tokens. This mask allows the final answer to integrate all parallel thoughts without violating autoregressive constraints.

A.4 Training Details

This section details the configuration used for supervised fine-tuning (SFT) of the large language model.

Dataset. For math reasoning, 3.5K of the problems are sampled from the Open-R1 (Hugging Face, 2025) filtered to include only those with more than 4 existing answer variations. We also randomly sample 1.5K from and DeepMath (He et al., 2025) dataset, which provides 3 answers per question, and 1.2K from s1k (Muennighoff et al., 2025) (0.4K filtered for clear answers) and limo (Ye et al., 2025) (0.8K full dataset). To enrich diversity, we use gpt-oss-20b (OpenAI, 2025) as a teacher model, generating additional solutions at temperature 0.8, yielding six reasoning paths per problem.

Parameter	Value
Batch Size	1
Gradient Accumulation Steps	8
Learning Rate	1×10^{-5}
Training Epochs	3
Context Length	28,672
Hardware	4 GPUs
Learning Rate Scheduler	Constant
Warmup Ratio	0.1
Weight Decay	0.05
Max Gradient Norm	0.5
Max Context Length	28K

Table 6: SFT Training Configuration for DeepSeek-R1-Distill-Qwen-1.5B

Parameter	Value
Batch Size	1
Gradient Accumulation Steps	4
Learning Rate	2×10^{-5}
Training Epochs	2
Context Length	28,672
Hardware	8 GPUs
Learning Rate Scheduler	Cosine with Minimum LR
Warmup Ratio	0.1
Weight Decay	0.05
Max Gradient Norm	0.5
Max Context Length	28K

Table 7: SFT Training Configuration for DeepSeek-R1-Distill-Qwen-7B

A.5 Evaluation settings

Our system is implemented using the vLLM inference framework (Kwon et al., 2023), integrated with our custom parallel generation engine. We employ Qwen-2.5-math (Yang et al., 2024) and lighteval framework (Habib et al., 2023) for evaluation of math and code reasoning tasks respectively. For the 1.5B parameter model, we employ a sampling temperature of 0.5 and a top-p value of 1.0, while for the 7B parameter model, we use a temperature of 0.6 and a top-p value of 1.0. To account for output randomness, we report pass@1 accuracy, calculated as $pass@1 = \frac{1}{k}\sum_{i=0}^{k} p_i$, where p_i is a binary indicator of correctness for the i-th response. Following DeepSeek-R1, we set k depending on the size of test dataset, thus we set $k = 16$ for AIME 2024, AIME 2025, AMC 2023, and $k = 4$ for MATH-500. For coding tasks, we use $k = 8$ samples per test input when computing pass@1.

A.6 Examples of training data

After sampling multiple answers from the teacher model, we construct the SFT training data as follows: (1) concatenate all sampled answers as separate reasoning paths; (2) insert special tokens to delineate the parallel reasoning paths; (3) append the ground-truth final answer. An example is shown in Figure 5.

	P=2	P=4	P=8
Last-Finish	32.1	37.1	42.5
Half-Finish	34.8	38.3	42.5
First-Finish (Default)	34.8	43.3	48.1

Table 8: Accuracy of ParaMind-1.5B on AIME 2024 under budget B for each reasoning path based on different strategies for terminating the parallel reasoning stage before proceeding to summarization.

	P=1	P=2	P=4	P=8
pass@1	51.9	57.1	63.3	68.8
maj@4	60.6	64.9	70.3	74.7
maj@8	68.8	68.9	72.1	77.6
maj@16	73.3	70.0	73.3	76.7

Table 9: ParaMind-7B together with majority voting on AIME 2024. P: number of parallel reasoning paths; $maj@k$: majority voting with k samples.

A.7 Examples of parallel answers

We illustrate the effectiveness of ParaMind through an example. Question: "*Jen enters a lottery by picking* 4 *distinct numbers from* $S = \{1, 2, 3, \cdots, 9, 10\}$. 4 *numbers are randomly chosen from* S. *She wins a prize if at least two of her numbers were* 2 *of the randomly chosen numbers, and wins the grand prize if all four of her numbers were the randomly chosen numbers. The probability of her winning the grand prize given that she won a prize is* $\frac{m}{n}$ *where* m *and* n *are relatively prime positive integers. Find* $m + n$. " ParaMind produces the following output (shortened for brevity) shown in Figure 6.

This example demonstrates that the generated parallel answers employ diverse reasoning strategies that complement each other. By considering multiple reasoning paths, ParaMind can mitigate the risk of being misled by a flawed initial line of thought.

A.8 Termination Strategies for the Parallel Reasoning Stage

We compare three strategies for terminating the parallel reasoning stage before proceeding to summarization: (1) *Last-Finish:* Wait for all P paths to complete. (2) *Half-Finish:* Terminate when $P/2$ paths have completed. (3) *First-Finish:* Terminate

when the first path completes (our default strategy).

As shown in Table 8, the First-Finish strategy yields the best performance. We attribute this to the fact that it maintains equal reasoning lengths across all paths, preventing any single path from dominating the context and ensuring a balanced contribution to the summarization stage. It is also, by definition, the most computationally efficient strategy.

A.9 Ablation

A.9.1 Ablation on Special Control Token and Thought Embeddings

We ablate the special control tokens (*e.g.* <think i>) and thought-specific positional embeddings in ParaMind on the AIME 2024 benchmark using the 1.5B model with $P = 4$ parallel paths. Table 10 presents the results. We find that without special tokens and thought embeddings, the accuracy of ParaMind drops 0.6%–3.1%, 1.4%–4.3% respectively, demonstrating the necessity of these designs for inducing thought diversity and resolving positional ambiguity during summarization.

A.9.2 Flattened Positional Encoding Ablation

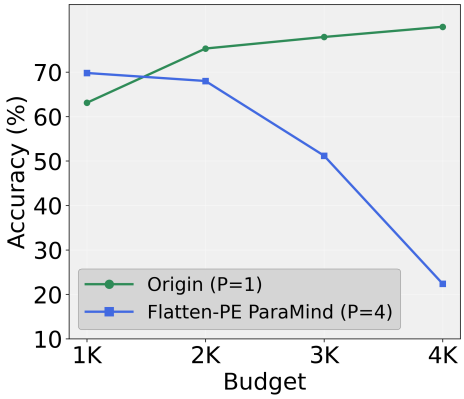


Figure 7: **Flattened PE Ablation:** Comparison on R1-1.5B trained with Flattened PE and original model (sequential) under Math-500 dataset.

We further ablate the use of flattened positional encoding (PE) as an alternative to our sequence-aware thought-specific PE, where unique absolute positions are assigned across all paths: $m = l_x + i \cdot l_{\max} + t$, with l_x as the input length, i indexing the reasoning path, and t the token position within that path. Figure 7 shows the performance degradation as P increases on AIME 2024. As illustrated, the Flatten-PE ParaMind achieves high accuracy under low token budgets (e.g., approximately 70% at 1K tokens) but experiences a rapid decrease as the budget increases to 4K tokens, dropping to around 30%. In contrast, the original sequential approach (P=1) shows steady improvement over the same budget range. While this resolves positional collisions, it results in large positional indices as P increases. Typical positional encoding mechanisms such as Rotary Position Embedding (RoPE) encodes relative positions via rotations, and large index differences $|m - n|$ cause attention scores to decay. As a result, tokens from earlier paths (*i.e.* lower i of $r^{(i)}$) contribute less when generating the final answer, introducing imbalance across paths. Our thought-specific PE, by contrast, maintains balanced attention through learnable segment identities, yielding consistent gains.

	P=2	P=4	P=8
ParaMind-1.5B	34.8	43.3	48.1
Thought Embedding Ablation	33.3	39.0	46.7
Special Token Ablation	34.2	40.2	45.2

Table 10: Ablation study on the effect of thought embedding (AIME 2024, avg@16, $t = 0.5$, $B = 16$K).

A.10 Why ParaMind Outperforms Majority Voting

Setup and notation. Let $(q, R, a) \sim \mathcal{D}$, where q is the input query, $R = (r^{(1)}, \ldots, r^{(P)})$ are P reasoning paths (token trajectories), and a is the true answer taking values in a finite set $\mathcal{A}$. Denote the data posterior by $p_{\text{data}}(a \mid q, R)$. Let $A = f(R) = (a^{(1)}, \ldots, a^{(P)})$ be the per-path answers extracted deterministically from R. An *aggregator* is any conditional distribution $p(a \mid q, R)$; we focus on two classes:

- **Voting-only aggregators:** $p(a \mid q, R)$ that depend on R only through A, i.e. $p(a \mid q, R) = p(a \mid q, A)$.
- **Path-aware aggregators:** $p(a \mid q, R)$ that may use the full R.

Markov chain view. The dependencies among these variables can be expressed as the Markov chain

$$q \longrightarrow R \longrightarrow A \longrightarrow a.$$

Since A is a deterministic function of R, and a depends on the reasoning process through R, this chain emphasizes that conditioning on A discards

part of the information about a contained in R. By the data-processing inequality,

$$I(a; R \mid q) \;\geq\; I(a; A \mid q),$$

with strict inequality whenever reasoning paths carry extra signal about a beyond the final extracted answers.

Information gap identity. The conditional mutual informations satisfy the chain rule

$$I(a; R \mid q) = I(a; A \mid q) + I(a; R \mid A, q).$$

Since $A = f(R)$ is deterministic, $I(a; R \mid A, q) \geq 0$. Moreover, the conditional mutual information $I(a; R \mid A, q)$ admits the following exact representation as an expected KL:

$$I(a; R \mid A, q) = \mathbb{E}_{(q,A)}\, \mathbb{E}_{R|q,A}\Big[D_{\mathrm{KL}}\big(p_{\mathrm{data}}(\cdot \mid q, R) \,\|p_{\mathrm{data}}(\cdot \mid q, A)\big)\Big].$$

Equivalently (taken as expectation over (q, R)),

$$I(a; R \mid q) - I(a; A \mid q) = \mathbb{E}_{(q,R)}\Big[D_{\mathrm{KL}}\big(p_{\mathrm{data}}(\cdot \mid q, R) \,\|p_{\mathrm{data}}(\cdot \mid q, A)\big)\Big].$$

We will call the left-hand side the *information gap*.

Optimal KL for voting-only aggregators. For any voting-only aggregator $p(a \mid q, A)$, the expected forward KL to the data posterior is

$$\mathbb{E}_{(q,R)}\Big[D_{\mathrm{KL}}\big(p_{\mathrm{data}}(\cdot \mid q, R) \,\|\, p(\cdot \mid q, A)\big)\Big].$$

For each (q, A), the choice of $p(\cdot \mid q, A)$ minimizing the inner expectation over $R \mid q, A$ is precisely $p_{\mathrm{data}}(\cdot \mid q, A)$. Therefore the minimal achievable expected KL over all voting-only aggregators equals the information gap:

$$\begin{aligned}
&\inf_{p(\cdot|q,A)} \mathbb{E}_{(q,R)}\Big[D_{\mathrm{KL}}\big(p_{\mathrm{data}}(\cdot \mid q, R) \,\|\, p(\cdot \mid q, A)\big)\Big] \\
&= \mathbb{E}_{(q,R)}\Big[D_{\mathrm{KL}}\big(p_{\mathrm{data}}(\cdot \mid q, R) \,\|\, p_{\mathrm{data}}(\cdot \mid q, A)\big)\Big] \\
&= I(a; R \mid q) - I(a; A \mid q).
\end{aligned}$$

Thus the information gap is an exact lower bound on the expected KL that any voting-only aggregator must incur. In fact, this bound is only attained by the Bayes-optimal choice $p(a \mid q, A) = p_{\mathrm{data}}(a \mid q, A)$, which requires full knowledge of the data distribution. Heuristic rules such as majority voting are far more restrictive and generally achieve strictly larger expected KL.

ParaMind (KL minimizer) vs. voting. Let $\mathcal{F} = \{p_\theta(a \mid q, R) : \theta \in \Theta\}$ be a family of path-aware aggregators (e.g., parameterized models trained with SFT on full paths). Define

$$\theta^\star = \arg\min_{\theta \in \Theta} \mathbb{E}_{(q,R)}\Big[D_{\mathrm{KL}}\big(p_{\mathrm{data}}(\cdot \mid q, R) \,\|\, p_\theta(\cdot \mid q, R)\big)\Big].$$

Two observations follow immediately:

1. If $\mathcal{F}$ contains the Bayes-optimal voting aggregator $p_{\mathrm{data}}(\cdot \mid q, A)$ as a special case (i.e., some $p_\theta(\cdot \mid q, R) = p_{\mathrm{data}}(\cdot \mid q, A)$ for all (q, R)), then by minimality of $\theta^\star$,

$$\begin{aligned}
&\mathbb{E}\big[D_{\mathrm{KL}}(p_{\mathrm{data}} \| p_{\theta^\star})\big] \leq \\
&\inf_{p(\cdot|q,A)} \mathbb{E}\big[D_{\mathrm{KL}}(p_{\mathrm{data}} \| p(\cdot \mid q, A))\big] \\
&= I(a; R \mid q) - I(a; A \mid q).
\end{aligned}$$

2. If $\mathcal{F}$ is sufficiently expressive to approximate $p_{\mathrm{data}}(\cdot \mid q, R)$ well, then the left-hand side above can be made small (approaching zero), whereas the right-hand side equals the information gap and is strictly positive whenever $I(a; R \mid A, q) > 0$.

Consequently, in any non-degenerate situation where intermediate reasoning tokens in R carry information about a beyond A, path-aware SFT (ParaMind) can attain strictly lower expected KL than any voting-only aggregator.

From KL to 0–1 risk. Define the expected classification error (0–1 risk) of an aggregator p by

$$\mathrm{TV}\big(p_{\mathrm{data}}(\cdot \mid q, R),\, p_{\theta^\star}(\cdot \mid q, R)\big) \leq \sqrt{\tfrac{1}{2}\, D_{\mathrm{KL}}\Big(p_{\mathrm{data}}(\cdot \mid q, R) \,\Big\|\, p_{\theta^\star}(\cdot \mid q, R)\Big)}.$$

Using the fact that the per-context increase in 0–1 risk is bounded by this TV distance, and taking expectation, we obtain

$$\mathcal{R}(p_{\theta^\star}) \leq \mathcal{R}(p_{\mathrm{data}}) + \sqrt{\tfrac{1}{2}\, \mathbb{E}_{(q,R)}\Big[D_{\mathrm{KL}}\big(p_{\mathrm{data}}(\cdot \mid q, R) \,\|\, p_{\theta^\star}(\cdot \mid q, R)\big)\Big]}.$$

Combining this with the KL comparison above gives the desired qualitative statement: when the information gap is positive, a path-aware aggregator that closely fits the data posterior will achieve lower 0–1 risk than any voting-only aggregator.

Remarks and caveats.

- The main identity linking mutual information and expected KL is exact and does not rely on asymptotics; it follows from the conditional mutual information representation $I(X; Y \mid Z) = \mathbb{E}_Z \mathbb{E}_{Y|Z} D_{\mathrm{KL}}(P_{X|Y,Z} \| P_{X|Z})$.

- Assumptions: we assumed a finite label alphabet for clarity (so KL and TV are finite); the same arguments extend to standard measurable settings with appropriate integrability conditions.

- Practical caveats: the inequalities above compare *best-possible* elements of model families. In practice, finite training data, limited model capacity, and optimization error mean $\theta^\star$ may not reach the theoretical minimum. Nonetheless, the direction of the inequality indicates when and why Path-aware SFT (ParaMind) should outperform majority voting.

- Pinsker's inequality is loose; for numerical guarantees one may replace it with bounds tailored to the label loss or use calibrated surrogate-loss analyses.

Connection to ParaMind design. ParaMind explicitly trains an aggregator to condition on full paths R, thereby directly targeting the KL objective minimized by $\theta^\star$ above. The identity and minimax reasoning explain why conditioning on full reasoning paths recovers information that voting discards.

SFT Data Example

<User>*Determine the number of solutions to the equation $z^{100} = 1 + i$ that lie in the third quadrant of the complex plane.* You FIRST think about the reasoning process as an internal monologue and then summarize the reasoning process to get the final answer. The summary process MUST BE enclosed within <summary> </summary> tags. The final answer MUST BE put in `\boxed{}`.<Assistant><think><vllm_pad>

<think1>
I need to solve $z^{100} = 1 + i$ and find how many solutions are in the third quadrant of the complex plane. First, I recall that complex numbers can be represented in polar form. So, I'll start by writing $1 + i$ in polar form ...

<think2>
Okay, so I need to find how many solutions of the equation $z^{100} = 1 + i$ are in the third quadrant. Hmm, let's start by recalling that complex equations like this one can often be solved using polar form and De Moivre's theorem. Let me jot down what I remember ...
</think2>

<think3>
Okay, so I need to figure out how many solutions of the equation $z^{100} = 1 + i$ lie in the third quadrant of the complex plane. Hmm, let's start by recalling that in the complex plane, the third quadrant is where both the real and imaginary parts of the complex number are negative. So any solution z must satisfy $\text{Re}(z) < 0$ and $\text{Im}(z) < 0$ ...

<think4>
I need to solve the equation $z^{100} = 1 + i$ and find how many solutions are in the third quadrant of the complex plane. First, I recall that complex numbers can be expressed in polar form, which might help with exponents. So, let $z = re^{i\theta}$, where r is the modulus and θ is the argument. Then $z^{100} = r^{100}e^{i100\theta}$ ...
</think4>

<summary>
By analyzing multiple reasoning processes above, I concluded that: The final answer is $\boxed{25}$.
</summary>

Figure 5: An example of SFT training data.

Answer Example

<think1>
Use the conditional probability formula

$$P(A|B) = \frac{P(A \cap B)}{P(B)},$$

where A is winning the grand prize (all four numbers correct), and B is winning a prize (at least two numbers correct). Calculate

$$P(A) = \frac{1}{\binom{10}{4}} = \frac{1}{210}.$$

Then calculate $P(B)$ as the probability of having at least two numbers correct, using the complement method:

$$P(B) = 1 - P(0) - P(1) = 1 - \frac{\binom{6}{4}}{210} - \frac{\binom{4}{1}\binom{6}{3}}{210} = \frac{115}{210} = \frac{23}{42}.$$

The final conditional probability is

$$\frac{\frac{1}{210}}{\frac{23}{42}} = \frac{1}{115},$$

hence $m + n = 116$.
</think1>

<think2>
calculate $P(B)$ directly as the probability of having at least two numbers correct:

$$\binom{4}{2}\binom{6}{2} + \binom{4}{3}\binom{6}{1} + \binom{4}{4}\binom{6}{0} = 115.$$

The conditional probability is $\frac{1}{115}$

<think3>
Directly compute the intersection to confirm:

$$P(A|B) = \frac{1}{115}.$$

<think4>
Use the hypergeometric distribution formula to calculate $P(B)$:

$$P(B) = \sum_{k=2}^{4} \frac{\binom{4}{k}\binom{6}{4-k}}{\binom{10}{4}} = \frac{115}{210},$$

<summary>
By analyzing multiple reasoning processes above, I concluded that: The final answer is $\boxed{116}$.
</summary>

Figure 6: Answer Example.