

# ParaThinker: Native Parallel Thinking as a New Paradigm to Scale LLM Test-time Compute

Hao Wen<sup>1,\*</sup>, Yifan Su<sup>1,\*,‡</sup>, Feifei Zhang<sup>1</sup>, Yunxin Liu<sup>1</sup>, Yunhao Liu<sup>2</sup>, Ya-Qin Zhang<sup>1</sup>, Yuanchun Li<sup>1,†</sup>

Source Code: https://github.com/MobileLLM/ParaThinker

Recent advances in Large Language Models (LLMs) have been driven by test-time compute scaling - a strategy that improves reasoning by generating longer, sequential thought processes. While effective, this approach encounters a significant bottleneck as computation increases, where further computation offers only marginal performance gains. We argue this ceiling is not an inherent limit of the model's capability but a flaw in the scaling strategy itself, a phenomenon we term "Tunnel Vision", where a model's imperfect initial steps lock it into a suboptimal reasoning path. To overcome this, we introduce a new scaling paradigm: native thought parallelism. We present ParaThinker, an end-to-end framework that trains an LLM to generate multiple, diverse reasoning paths in parallel and synthesize them into a superior final answer. By exploring different lines of thoughts simultaneously, ParaThinker effectively sidesteps the Tunnel Vision issue and unlocks the model's latent reasoning potential. Our approach demonstrates that scaling compute in parallel (width) is a more effective and efficient way to superior reasoning than simply scaling sequentially (depth). On challenging reasoning benchmarks, ParaThinker achieves substantial accuracy improvements over sequential LLMs (12.3% for 1.5B and 7.5% for 7B models on average with 8 parallel paths), while adding only negligible latency overhead (7.1%). This enables smaller models to surpass much larger counterparts and establishes parallel thinking as a critical, efficient dimension for scaling future LLMs.

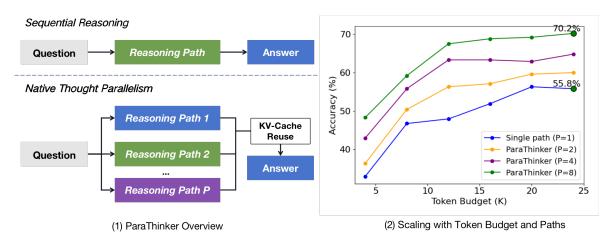


Figure 1 | Sequential vs. Parallel reasoning with ParaThinker framework and scaling results. (1) Illustrations of ParaThinker. (2) Parallel scaling results of ParaThinker-7B on AIME 2024 with varying numbers of reasoning paths (*P*). "Token budget" refers to the maximum token length allowed per reasoning path.

<sup>&</sup>lt;sup>1</sup>Institute for AI Industry Research (AIR), Tsinghua University

<sup>&</sup>lt;sup>2</sup>Global Innovation Exchange & Department of Automation, Tsinghua University

<sup>\*</sup>Equal contribution.

<sup>†</sup>Corresponding author: Yuanchun Li (liyuanchun@air.tsinghua.edu.cn).

<sup>&</sup>lt;sup>‡</sup>Work done during internships at AIR, Tsinghua University.

#### 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., 2025; 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 a fundamental question: Does the test-time scaling bottleneck stem from the inherent limitations of the model's capability, or from the imperfect test-time compute strategy? The answer to this question is important for understanding the bottleneck of test-time scaling. Our findings reveal that, given a fixed decoding token budget, the conventional self-refinement reasoning paradigm (adopted in state-of-the-art reasoning models like o1 and R1) constantly converges at a low accuracy that can be achieved with other simple scaling strategies (e.g., majority voting) under the same token budget. This suggests that the model's underlying capability is not the primary bottleneck; rather, the way we orchestrate test-time compute can be improved.

Through a closer analysis of the reasoning process in LLMs, 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 insights, we argue that the reasoning process of LLMs should be executed in a parallel, multi-threaded manner. By ensuring each thinking thread operates independently, we can mitigate Tunnel Vision and foster a diversity of thought. Furthermore, parallel thinking offers significant 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, ParaThinker, 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 developing ParaThinker 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. Specifically, our solution features three core innovations:

• **Specialized Control Tokens:** We introduce a set of trainable tokens (*e.g.*, <think i>) to explicitly guide the model's generation. Each <think i> token prompts the model to initiate a distinct reasoning path, which ensures diversity in reasoning.

- Thought-Specific Positional Embedding: To resolve positional ambiguity when merging parallel thoughts, we augment the standard positional encoding with a unique, learnable embedding for each reasoning path. This allows the model to unambiguously differentiate the origin of each token during the final summarization stage.
- SFT Training Pipeline: We employ a scalable supervised fine-tuning (SFT) strategy where the model is trained on reasoning paths sampled from a teacher model. By randomly assigning the specialized <think i> tokens during this process, the model learns to generalize, enabling it to generate more parallel paths at inference time than were seen during training.

We evaluate ParaThinker on challenging reasoning benchmarks: AIME 2024, AIME 2025, AMC 2023, and MATH-500 (Hendrycks et al., 2021), against baselines such as standard autoregressive reasoning (DeepSeek-AI, 2025), majority voting (Chen et al., 2024a), and re-prefilling. Our approach demonstrates a remarkable leap in performance, achieving significantly improved accuracy with slight latency overhead. This efficiency allows smaller LLMs equipped with our native thought parallelism to outperform much larger, standard reasoning models, charting a new path for scaling test-time compute.

In summary, the contributions of our work are: (1) We characterize the test-time scaling bottleneck in LLM reasoning and attribute it to a narrow reasoning pathway, termed Tunnel Vision, which restricts the model's exploration during generation. (2) We propose and demonstrate that thought parallelism is a better way to scale LLM test-time compute. (3) We introduce an end-to-end solution to enable native parallel thinking. The resulting model, ParaThinker, achieves higher accuracy than sequential LLMs by 12.3% and 7.5% for 1.5B and 7B models, respectively. Compared with majority voting, ParaThinker further improves accuracy by 4.3% and 2.0%.

# 2. Understanding the Scaling Bottleneck

To empirically ground our investigation, we first characterize the limitations of conventional test-time scaling. We identify a test-time scaling bottleneck that suggests a fundamental inefficiency in sequential reasoning (Section 2.1). Then, we diagnose its underlying cause, a phenomenon we term Tunnel Vision (Section 2.2). Then, we propose parallel scaling as an effective and efficient solution to it (Section 2.3).

#### 2.1. Is the Bottleneck Due to Limited LLM Capability or Suboptimal Scaling Strategy?

We examine the performance bottleneck in reasoning LLMs 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 (
 (
 A (A (A (A (A )). We also evaluate majority voting (Chen et al., 2024a) over A (A (A )) are allel samples, with each sample allocated A (A ) tokens. For clarity, we plot only A (A ) and A = 8 results in the accuracy-budget curves, and report the maximum accuracy obtained with A = 64. The results, shown in Figure 2a, demonstrate that the performance of a single reasoning path (green line) quickly meets 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., 2025; Fan et al., 2025; Cuadron et al., 2025). Different from

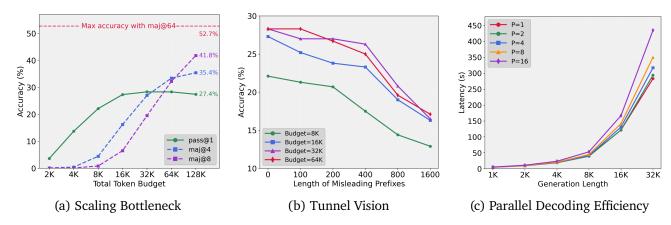


Figure 2 | Diagnosing the limitations of sequential reasoning and the potential of parallelism. All experiments use DeepSeek-R1-Distill-Qwen-1.5B on the AIME 2024 benchmark. (a) Scaling Bottleneck: Accuracy against the total number of token budget (for majority voting *e.g.*, maj@4, the total token budget is the sum across all parallel paths). (b) Tunnel Vision: Ability to recover from its own potential mistakes with different lengths of misleading prefixes. The model generates solutions starting from flawed prefixes of length  $n \in \{0, 100, ..., 1600\}$ , denoting the first n tokens of reasoning paths from the same model that previously resulted in a wrong answer. (c) Parallel Decoding Efficiency: Latency taken to decode  $P \in \{1, 2, 4, 8, 16\}$  parallel paths, each of length  $n \in \{1K, ..., 32K\}$ .

these approaches, we investigate the fundamental cause of this bottleneck to determine how to further bootstrap LLM test-time scaling. And results in Figure 2a 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 significant gap suggests that the bottleneck is not a hard limit of the model's reasoning capacity, but rather a symptom of the suboptimal test-time scaling strategy. Simply allocating more test-time compute to a single-sequence LLM is not as effective as exploring 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 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 2b, show a clear negative correlation: the longer the erroneous prefix, the lower the final accuracy. This indicates that the scaling bottleneck is a direct symptom of Tunnel Vision, where flawed initial tokens 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.

# 2.3. Native Thought Parallelism: An Effective and Efficient Solution

Given the limitations of sequential reasoning exposed by Tunnel Vision, exploring multiple lines of thought in parallel is a promising solution. While methods like majority voting (Chen et al., 2024a) validate the benefit of parallelism, they are primarily applicable to tasks with easily quantifiable or verifiable outputs (e.g., multiple-choice questions, numerical answers). Their utility is limited for more complex, open-ended domains (e.g., complex agentic workflows, coding, or mathematical proofs).

This necessitates a native parallel framework—one that enables an LLM to generate and integrate multiple reasoning paths within a single, end-to-end forward pass. Such a system should scale with the number of parallel paths while maintaining high computational efficiency. Previous approaches to parallel computation typically rely on external verifiers for search (Snell et al., 2024; Ghosal et al., 2025), which introduces a scalability bottleneck. Recent methods such as architectural parallelism (Chen et al., 2025a), still generate tokens sequentially, leaving them vulnerable to the same phenomenon, Tunnel Vision.

The efficiency of native thought parallelism stems from its hardware-friendly nature. The primary bottleneck in LLM decoding speed is typically memory access (parameter load, KV load/store), not raw computation (Sadhukhan et al., 2025). By generating P parallel reasoning paths simultaneously, we increase the computational workload for each memory access, improving the arithmetic intensity and better utilizing the GPU's processing power. We test the efficiency of parallel generation in Figure 2c, where we record the inference time of DeepSeek-R1-Distill-Qwen-1.5B based on vLLM framework (Kwon et al., 2023) on a single A800 GPU with different generation lengths, using a batch size of one. For a small number of parallel paths, generating L tokens for every path takes nearly the same amount of time as generating L tokens for a single path. Remarkably, even when decoding 16 parallel paths of length L, the total latency is less than  $2\times$  of decoding a single path of the same length. This hardware-level efficiency makes parallel exploration a practical and scalable strategy for overcoming Tunnel Vision and unlocking superior reasoning performance.

# 3. Model Design

As shown in Figure 3, ParaThinker generates an answer for a given question in two stages. In the parallel reasoning stage, ParaThinker generates multiple independent reasoning trajectories. In the summarization stage, it analyzes these diverse reasoning paths and then fuses them into a final answer. ParaThinker utilizes intermediate KV-cache representations from the reasoning stage, which eliminates the need for costly re-prefilling during summarization.

#### 3.1. Preliminaries

We denote an LLM by  $\pi_{\theta}$ , where  $\theta$  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  $y = (y_1, y_2, \dots, y_L)$  with the conditional probability:

$$\pi_{\theta}(y|x) = \prod_{t=1}^{L} \pi_{\theta}(y_t|x, y_{< t})$$
 (1)

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

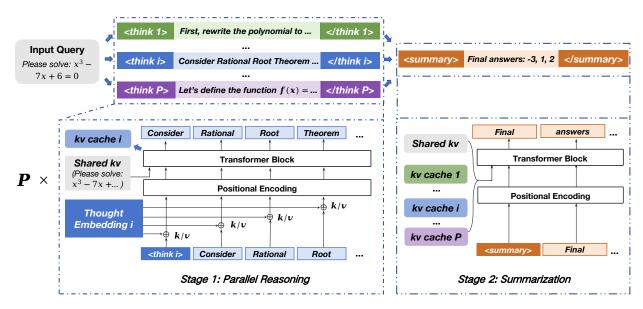


Figure 3 | ParaThinker architecture. For an input question, ParaThinker processes it in two stages: (1) Parallel Reasoning: ParaThinker generates *P* reasoning paths in parallel, guided by special <think i> tokens to encourage diverse thoughts, and employs thought embeddings to separate paths and prevent positional ambiguity; (2) Summarization: ParaThinker merges the reasoning paths by reusing their KV-caches to generate the final answer.

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.

#### 3.2. ParaThinker Workflow

As shown in Figure 3, 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:

$$\pi_{\theta}(r^{(i)}|x) = \prod_{t=1}^{L_i} \pi_{\theta}(r_t^{(i)}|x, s^{(i)}, r_{< t}^{(i)})$$
(2)

Here,  $s^{(i)}$  is a special control token that helps initiate a distinct reasoning path, which will be detailed in Section 3.3. 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  $\mathcal{R} = (r^{(1)}, r^{(2)}, \dots, r^{(p)})$  be the concatenation of all generated reasoning paths. The final answer a is then sampled from the model as follows:

$$\pi_{\theta}(a|x) = \prod_{t=1}^{La} \pi_{\theta}(a_t|x, \mathcal{R}, a_{< t})$$
(3)

Crucially, ParaThinker 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.

## 3.3. Special Tokens for Boosting Thought Diversity

ParaThinker 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  $\pi_{\theta}(r^{(i)}|x) = \prod_{t=1}^{L_i} \pi_{\theta}(r^{(i)}_t|x,s^{(i)},r^{(i)}_{< t})$  will be conditioned by  $s^{(i)}$ . The closing 
 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 is a simple yet powerful mechanism to guide the model's generation process towards diverse and parallel lines of thought.

# 3.4. 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  $r^{(i)}$ ) share identical positional encodings. This causes 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:  $m = l_x + i \cdot l_{\text{max}} + t$ , 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 generation tasks, ParaThinker 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:

$$\tilde{k}_t^{(j)} = R_t(k_t^{(j)} + T^{(j)}) \tag{4}$$

$$\tilde{v}_t^{(j)} = v_t^{(j)} + T^{(j)} \tag{5}$$

Here,  $l_{max}$  denotes the maximum token number for each reasoning path, and  $R_t$  is the corresponding RoPE rotation matrix. 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:

$$score(n, m) = (R_n q_n^{(i)})^T \tilde{k}_m^{(j)} = (R_n q_n^{(i)})^T [R_m (k_m^{(j)} + T^{(j)})]$$
(6)

Using the RoPE property  $(R_n)^T R_m = R_{m-n}$ , Eq. 6 can be simplified into two distinct components:

$$score(n, m) = \underbrace{q_n^T R_{m-n} k_m^{(j)}}_{Content-to-Content} + \underbrace{q_n^T R_{m-n} T^{(j)}}_{Content-to-Segment}$$
(7)

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, learned thought embedding, this term provides an unambiguous signal for the model to differentiate between parallel streams of text, solving the positional ambiguity.

# 4. Training and Deployment of ParaThinker

We use SFT to train ParaThinker. The special challenges include: (1) How to let LLMs know the meanings of <think i> tokens and generate diverse reasoning paths during training? (Section 4.1) (2) How to design attention mask for ParaThinker training? (Section 4.2) (3) How to design inference engine of ParaThinker? (Section 4.3)

#### 4.1. Scalable Train Data Curation

To effectively enable native parallel reasoning, 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:  $\hat{y} = (\sinh 1)r^{(1)} < \sinh 1 > \ldots, (\sinh \hat{P})r^{(\hat{P})} < \sinh \hat{P} > \sinh 2r^{(\hat{P})} > \ldots$ . The resulting  $Data_{r-1}^{\hat{P}} = (x, \hat{y})$  pairs are then used for SFT.

**Extensible 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  $Data_{sft}$ , that is,  $\hat{P}$  is less than P in many cases. Thus, during SFT stage, LLMs have to learn to extend to  $(r^{(\hat{P}+1)}, \ldots, r^{(P)})$  with training data  $Data_{r=1}^{\hat{P}}$ . Therefore, we develop a dynamic special token sampling method for extensible special tokens training. Specifically, at each training batch, we randomly sample  $\hat{P}$  special tokens from  $\{1, \ldots, P\}$ . These sample  $\hat{P}$  tokens are placed at the beginning of each reasoning sequence, and the special tokens embeddings are trainable. Thus the special token embeddings are diverse after enough training, and lead LLMs to different reasoning directions.

#### 4.2. Attention Mask Design

To enable effective parallel reasoning, ParaThinker 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:  $A_{i,j} = \operatorname{Softmax}\left(\frac{q_i \cdot k_j + M_{i,j}}{\sqrt{d_k}}\right)$ . The attention mask for the i-th reasoning path  $(r^{(i)})$  is defined as:

$$M_{t,j}^{r^{(i)}} = \begin{cases} 0, & \text{if } j \le t \text{ and } j \in \{1, \dots, l_x\} \cup \text{Ind}_i \\ -\infty, & \text{otherwise} \end{cases}$$
 (8)

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:

$$M_{t,j}^{A} = \begin{cases} 0, & \text{if } j \le t \text{ and } j \in \{1, \dots, l_x\} \cup \bigcup_{i=1}^{p} \text{Ind}_i \cup \text{Ind}_a \\ -\infty, & \text{otherwise} \end{cases}$$
 (9)

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.

# 4.3. Inference Engine

The inference engine for ParaThinker is built upon the vLLM framework (Kwon et al., 2023) to leverage its efficient parallel scaling. 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 synchronous decoding phase terminates for all paths simultaneously as soon as the one of the following conditions is met: (1) Path Completion: Any one of the *P* paths generates an end-of-sequence (EOS) token. (2) Budget Exceeded: Any one of the *P* paths reaches the predefined token budget, B. This uniform termination strategy ensures all reasoning paths maintain an equal length, preventing processing imbalance. As empirically justified in Section 5.4, this approach 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. The final answer is then autoregressively decoded from this unified context.

# 5. Experiment

In this section, we conduct experiments to address the following research questions:

- Scalability and Performance: How does ParaThinker's reasoning accuracy scale with an increase in parallel reasoning paths number and generation budget? (Section 5.2)
- **Inference Efficiency:** What are the trade-offs in terms of inference latency and memory consumption as we scale the number of parallel paths? (Section 5.3)
- **Fine-Grained Analysis:** What is the optimal termination strategy for parallel reasoning stage, and how much does each component contribute? (Section 5.4 and Section 5.5)

## 5.1. Experimental Setup

**Training Details:** Our experiments are based on a Qwen-2.5 (Qwen et al., 2025) 1.5B and 7B parameter model distilled from DeepSeek-R1 (DeepSeek-AI, 2025), which we denote as original models below.

For SFT, we construct a parallel reasoning dataset with 6.2K problem-solution pairs. 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. Each instance thus consists of a query ( $x_i$ ), ground-truth answer ( $a_i$ ), and  $\hat{P} = 6$  distinct reasoning paths.

During fine-tuning, we use a maximum context length of 28K tokens. The model is trained for 2-3 epochs on multiple A800 GPUs. More training details can be found in Appendix A.1. During every step, we randomly choose a path number P from the set  $\{2,4,6\}$  and construct a training sample by concatenating P samples, which is detailed in Appendix A.2.

For data ablation, we ensure comparable dataset volume by unrolling our parallel dataset into  $\tilde{D}_{ablation} = \{\{x_i, (r_i^{(j)}, a_i)\}_{j=1}^{\hat{p}}\}_{i=1}^{|D|}$ , then fine-tune the 1.5B baseline with identical settings. We then report results for *sequential*, *majority voting*, and *re-prefilling* against ParaThinker-1.5B.

**Baselines:** We compare ParaThinker against:

- Sequential: Direct reasoning with original 1.5B/7B models.
- Majority Voting: Generate P independent paths and return the majority answer (Chen et al., 2024a).
- *Re-Prefilling:* Generate *P* paths, concatenate them, and feed the full context into the model for summarization. This mimics ParaThinker's summarization but is inefficient since KV caches are not reused (Section 5.3).

**Benchmarks and Evaluation Setup:** We evaluate our model on four challenging mathematical reasoning benchmarks: AIME 2024, AIME 2025, AMC 2023, and MATH-500 (Hendrycks et al., 2021). We employ two primary criteria for evaluation: effectiveness, measured by accuracy; and efficiency, measured by performance under a fixed token budget. To assess efficiency, we use a budget control method where each reasoning path is limited to a maximum of B tokens ( $|r^{(i)}| \leq B$ ). If a model reaches this budget without naturally stopping, we enforce termination and then initiate the summarization stage by adding the (<summary>) token. This allows us to examine the utilization of the test-time scaling budget.

Our system is implemented using the vLLM inference framework (Kwon et al., 2023), integrated with our custom parallel generation engine. 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.

	AIME 2024	AIME 2025	AMC 2023	MATH-500	Average
Original Model: DeepSeek-R1-distill-Qwen-1.5B					
Sequential (16K)	26.1	22.4	67.1	81.2	49.2
Sequential (32K)	28.3	24.5	68.9	81.8	50.9
Sequential (64K)	27.1	25.5	67.7	81.7	50.5
Sequential (128K)	27.4	22.1	68.0	81.8	49.8
Majority (2×16K)	25.9	23.0	67.0	81.4	49.3
Majority (4×16K)	32.9	27.5	74.3	86.7	55.4
Majority (8×16K)	41.0	31.8	79.8	89.0	60.4
Reprefill (2×16K)	30.4	26.7	70.6	60.8	47.1
Reprefill (4×16K)	24.2	25.8	61.3	58.6	42.5
Reprefill (8×16K)	14.2	13.3	60.0	55.3	35.7
ParaThinker-1.5B (2×16K)	34.8	24.2	73.1	87.5	54.9
ParaThinker-1.5B (4×16K)	43.3	26.7	80.8	88.7	59.9
ParaThinker-1.5B (8×16K)	48.1	31.9	83.1	89.7	63.2
Original Model: DeepSeek-R1-	distill-Qwen-7B				
Sequential (16K)	51.9	37.9	88.4	91.2	67.4
Sequential (32K)	55.5	37.9	89.8	92.0	68.8
Sequential (64K)	56.0	39.6	89.8	92.5	69.5
Sequential (128K)	52.7	40.4	89.8	92.6	68.9
Majority (2×16K)	52.3	38.3	88.4	91.4	67.6
Majority (4×16K)	60.6	43.1	92.2	93.5	72.4
Majority (8×16K)	68.8	49.6	93.1	94.2	76.4
Reprefill (2×16K)	42.9	33.8	88.1	63.8	57.2
Reprefill (4×16K)	43.3	33.3	86.3	63.2	56.5
Reprefill (8×16K)	43.3	31.7	91.9	63.7	57.7
ParaThinker-7B (2×16K)	57.1	46.0	89.5	93.2	71.5
ParaThinker-7B (4×16K)	63.3	46.9	91.7	94.2	74.0
ParaThinker-7B (8×16K)	68.8	51.3	93.3	94.5	77.0

Table 1 | Accuracy of ParaThinker and baselines on MATH-500, AIME 2024, AIME 2025, and AMC 2023. We report Pass@1 accuracy (%). Values in brackets indicate the maximum generation length L (e.g., 16K); for parallel generation methods, we use  $P \times L$  to denote generating P reasoning paths, each with a maximum length of L.

#### 5.2. Scalability and Performance

Table 1 compares ParaThinker with baseline methods under different token budgets. Compared with sequential LLMs, ParaThinker 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 budget on average (e.g.,  $2 \times 16$ K vs. 32K), demonstrating the effectiveness of parallel reasoning. On average, ParaThinker achieves 4.3% and 2.0% higher accuracy than majority voting on the 1.5B and 7B models, respectively. This indicates that the summarization stage captures a richer aggregation strategy than vote counting. Besides, for P = 2, ParaThinker substantially outperforms maj@2 (which randomly selects one result from two reasoning paths and thus closely resembles sequential reasoning), showing that it does more than simply pick the

	B = 8K	B=16K	B = 32K	B = 64K	B = 128K
P = 1	23.5	26.1	28.3	27.1	27.4
P = 2	27.1	29.2	34.8	35.8	25.0
P = 4	18.1	30.2	38.1	43.3	36.7
P = 8	7.9	22.3	35.0	41.5	48.1

Table 2 | Accuracy of baseline (P = 1) and ParaThinker-1.5B (P = 2/4/8) on AIME 2024 with different reasoning path number P and total token budget B (total token length of all reasoning paths).

	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 | ParaThinker-1.5B together with majority voting on AIME 2024. *P*: number of parallel reasoning paths; *maj@k*: majority voting with *k* samples.

majority answer and instead learns to integrate information across reasoning paths. By contrast, the performance of the re-prefilling baseline degrades with more paths, which we attribute to the context length limitations of the flattened positional encoding scheme discussed in Section 3.4.

We analyze how performance scales with the number of parallel paths in Table 2, 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 ParaThinker continues to improve. These results indicate that ParaThinker effectively extends the scaling law beyond the point where sequential reasoning models typically encounter a test-time scaling bottleneck.

We further analyze the relationships between ParaThinker and majority voting in detail. The result is shown in Table 3. We find that our method does not conflict with the majority voting, ParaThinker with majority voting can achieve higher accuracy than only using ParaThinker. The highest accuracy of ParaThinker-1.5B+maj@8 can achieve accuracies of 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. Besides, majority voting is not able to be applied to scenarios where results can't be quantified (*e.g.*, coding, document generation, etc.), which we will evaluate ParaThinker as a future work.

#### **5.3.** Inference Efficiency

A key advantage of ParaThinker is its efficiency. Figure 4 shows that latency does not grow linearly with the number of paths *P*, and with *P* grows, the total inference latency increases slightly. 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. The inference latency of ParaThinker even decreases slightly under some budgets when number of parallel size grows, because we terminate the reasoning process once the first reasoning path stops, and when the number of paths increases, the probability of ParaThinker terminating earlier grows larger. The efficiency of our method gives us the proof that we can achieve greater accuracy through parallel scaling within acceptable inference latency.

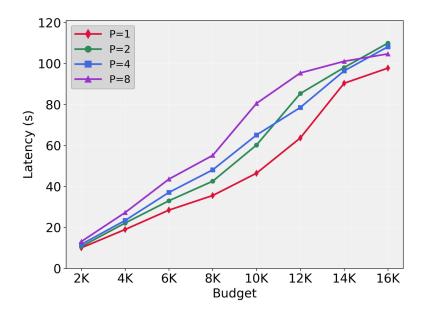


Figure 4 | Total latency of ParaThinker-1.5B (batch size=1) with different number of parallel paths (P) under different generation budgets for each path (i.e., total decoding latency of  $P \times B$  tokens for ParaThinker with P paths.

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

Table 4 | Accuracy of ParaThinker-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.

## 5.4. 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 4, 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.

#### 5.5. Ablation Study

To have a fine-grained analysis of our key design choices, we conduct an ablation study on AIME 2024 based on ParaThinker-1.5B.

• Train Data: We test whether the performance gain of ParaThinker comes from the training dataset.

	AIME 2024	AIME 2025	AMC 2023	MATH-500	Average
DeepSeek-R1-distill-Qwen-1.5	3-SFT				
Sequential (16K)	26.3	18.5	66.0	81.1	48.0
Sequential (32K)	22.9	22.1	64.1	77.6	46.7
Sequential (64K)	25.8	17.3	62.2	77.6	45.7
Sequential (128K)	24.8	21.9	63.6	78.6	47.2
Majority (2×16K)	26.0	18.1	66.3	81.0	47.9
Majority (4×16K)	32.2	23.4	72.1	86.5	53.6
Majority (8×16K)	42.5	27.1	79.8	89.2	59.7
Reprefill (2×16K)	23.3	16.3	65.6	76.8	45.5
Reprefill (4×16K)	15.0	11.7	55.6	70.6	38.2
Reprefill (8×16K)	15.8	9.2	58.8	66.6	37.6
ParaThinker-1.5B					
ParaThinker-1.5B (2×16K)	34.8	24.2	73.1	87.5	54.9
ParaThinker-1.5B (4×16K)	43.3	26.7	80.8	88.7	59.9
ParaThinker-1.5B (8×16K)	48.1	31.9	83.1	89.7	63.2

Table 5 | Accuracy of ParaThinker and baselines on MATH-500, AIME 2024, AIME 2025, and AMC 2023. We report Pass1 accuracy (%). Values in brackets indicate the maximum generation length L (e.g., 16K); for parallel generation methods, we use  $P \times L$  to denote generating P reasoning paths, each with a maximum length of L.

	P=2	P=4	P=8
ParaThinker-1.5B	34.8	43.3	48.1
Thought Embedding Ablation	33.3	39.0	46.7

Table 6 | Ablation study on the effect of thought embedding (AIME 2024, avg@16, t = 0.5, B = 16K).

The training details of data ablation are stated in Section 5.1, where we use all the data from our dataset (6 samples for each question) to finetune the original LLM, remaining all other settings the same. Table 5 indicates that finetuning does not improve performance, with results even slightly worse than the original LLM. ParaThinker, on the other hand, outperforms all baselines across budgets, confirming its effectiveness.

• Thought Embedding: We conduct two ablation studies of thought embedding: (1) removing the embedding entirely and (2) replacing it with a naive flattened positional encoding as stated in Section 3.4. As shown in Table 6, the flattened encoding leads to a severe accuracy drop, especially with larger budgets, confirming the detrimental effect of long-range positional decay. Surprisingly, removing the embedding altogether, while still underperform our proposed method, is superior to the flattened approach. This suggests the model can partially infer path distinctions from context but is misled by the ambiguous signals of a naive long-range positional encoding.

#### 6. Related Work

#### 6.1. 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., 2025; 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 & 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.

#### 6.2. 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 & 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), 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.

## 6.3. 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., 2025; 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. Other works (Yang et al., 2025b; Pan et al., 2025; Rodionov et al., 2025; Jin 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. ParaThinker, 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.

### 7. Conclusion

Our work identifies a fundamental limitation of current test-time compute strategies, where sequential reasoning gets caught in Tunnel Vision set by its own initial tokens. We demonstrate that this is not an inherent model capability limit but a failure of the scaling strategy itself. We then introduce ParaThinker, a framework for native parallel reasoning that sidesteps these traps by generating and synthesizing multiple thought-paths simultaneously. This approach proves that scaling compute in parallel (width) is a more effective and efficient path to superior reasoning than simply scaling sequentially (depth). While our method is a significant first step, future work could explore more advanced aggregation strategies and deeper reinforcement learning. Fundamentally, our results argue that to unlock the next level of reasoning, LLM test-time compute should scale in parallel.

#### References

Marianne Arriola, Subham Sekhar Sahoo, Aaron Gokaslan, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Justin T Chiu, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tyEyYT267x.

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

Lingjiao Chen, Jared Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. Are more llm calls all you need? towards the scaling properties of compound ai systems. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), Advances in Neural Information Processing Systems, volume 37, pp. 45767–45790. Curran Associates, Inc., 2024a. URL https://proceedings.neurips.cc/paper\_files/paper/2024/file/51173cf34c5faac9796a47dc2fdd3a71-Paper-Conference.pdf.

Mouxiang Chen, Binyuan Hui, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Jianling Sun, Junyang Lin, and Zhongxin Liu. Parallel scaling law for language models, 2025a. URL https://arxiv.org/abs/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. Do not think that much for 2+3=? on the overthinking of o1-like llms, 2025b. URL https://arxiv.org/abs/2412.21187.

- Xinyun Chen, Ryan Andrew Chi, Xuezhi Wang, and Denny Zhou. Premise order matters in reasoning with large language models. In *Forty-first International Conference on Machine Learning*, 2024b. URL <a href="https://openreview.net/forum?id=4zAHgkiCQg">https://openreview.net/forum?id=4zAHgkiCQg</a>.
- 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. The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks, 2025. URL https://arxiv.org/abs/2502.08235.
- DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.
- Chenrui Fan, Ming Li, Lichao Sun, and Tianyi Zhou. Missing premise exacerbates overthinking: Are reasoning models losing critical thinking skill?, 2025. URL https://arxiv.org/abs/2504.06514.
- Guhao Feng, Yihan Geng, Jian Guan, Wei Wu, Liwei Wang, and Di He. Theoretical benefit and limitation of diffusion language model, 2025. URL https://arxiv.org/abs/2502.09622.
- Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *CoRR*, abs/2502.05171, February 2025. URL https://doi.org/10.48550/arXiv.2502.05171.
- Soumya Suvra Ghosal, Souradip Chakraborty, Avinash Reddy, Yifu Lu, Mengdi Wang, Dinesh Manocha, Furong Huang, Mohammad Ghavamzadeh, and Amrit Singh Bedi. Does thinking more always help? understanding test-time scaling in reasoning models, 2025. URL https://arxiv.org/abs/2506.04210.
- Google. Gemini 2.5: Our most intelligent ai model. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking, March 2025.
- Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rstar-math: Small llms can master math reasoning with self-evolved deep thinking, 2025. URL https://arxiv.org/abs/2501.04519.
- Zhengfu He, Tianxiang Sun, Qiong Tang, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. DiffusionBERT: Improving generative masked language models with diffusion models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4521–4534, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.248. URL https://aclanthology.org/2023.acl-long.248/.
- Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, et al. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*, 2025.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

- Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL https://github.com/huggingface/open-r1.
- Tian Jin, Ellie Y. Cheng, Zack Ankner, Nikunj Saunshi, Blake M. Elias, Amir Yazdanbakhsh, Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding, 2025. URL https://arxiv.org/abs/2502.11517.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.
- Bespoke Labs. Bespoke-stratos: The unreasonable effectiveness of reasoning distillation. www.bespokelabs.ai/blog/bespoke-stratos-the-unreasonable-effectiveness-of-reasoning-distillation, 2025. Accessed: 2025-01-22.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.
- Zheng Li, Qingxiu Dong, Jingyuan Ma, Di Zhang, and Zhifang Sui. Selfbudgeter: Adaptive token allocation for efficient llm reasoning. *arXiv preprint arXiv:2505.11274*, 2025.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/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. s1: Simple test-time scaling. arXiv preprint arXiv:2501.19393, 2025.
- OpenAI. Openai o1 system card. 2024. URL https://arxiv.org/abs/2412.16720.
- OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.
- Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models, 2025. URL https://arxiv.org/abs/2504.15466.
- Xiangyu Qi, Ashwinee Panda, Kaifeng Lyu, Xiao Ma, Subhrajit Roy, Ahmad Beirami, Prateek Mittal, and Peter Henderson. Safety alignment should be made more than just a few tokens deep. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6Mxhg9PtDE.
- Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement fine-tuning, 2025. URL https://arxiv.org/abs/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, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.
- Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazarian, Anton Sinitsin, Denis Kuznedelev, and Dan Alistarh. Hogwild! inference: Parallel llm generation via concurrent attention, 2025. URL https://arxiv.org/abs/2504.06261.
- Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Representation Learning*, volume 2025, pp. 6835–6850, 2025. URL https://proceedings.iclr.cc/paper\_files/paper/2025/file/13f972adf12bdf886583d48cd528002f-Paper-Conference.pdf.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.
- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.
- Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.
- Yi Sun, Han Wang, Jiaqiang Li, Jiacheng Liu, Xiangyu Li, Hao Wen, Yizhen Yuan, Huiwen Zheng, Yan Liang, Yuanchun Li, and Yunxin Liu. An empirical study of llm reasoning ability under strict output length constraint. *arXiv preprint arXiv:2504.14350*, April 2025. doi: 10.48550/arXiv.2504.14350.
- Kimi Team et al. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL https://arxiv.org/abs/2501.12599.
- Guanghan Wang, Yair Schiff, Subham Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. *arXiv preprint arXiv:2503.00307*, 2025.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.
- Hao Wen, Xinrui Wu, Yi Sun, Feifei Zhang, Liye Chen, Jie Wang, Yunxin Liu, Ya-Qin Zhang, and Yuanchun Li. Budgetthinker: Empowering budget-aware llm reasoning with control tokens. *arXiv preprint arXiv:2508.17196*, 2025.
- Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009. ISSN 0001-0782. doi: 10.1145/1498765.1498785. URL https://doi.org/10.1145/1498765.1498785.

- Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. In Jian Su, Kevin Duh, and Xavier Carreras (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1296–1306, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1137. URL https://aclanthology.org/D16-1137/.
- Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models, 2025. URL https://arxiv.org/abs/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. Redstar: Does scaling long-cot data unlock better slow-reasoning systems?, 2025. URL https://arxiv.org/abs/2501.11284.
- Rongwu Xu, Zehan Qi, and Wei Xu. Preemptive answer "attacks" on chain-of-thought reasoning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 14708–14726, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.876. URL https://aclanthology.org/2024.findings-acl.876/.
- An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.
- Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. Multiverse: Your language models secretly decide how to parallelize and merge generation. *arXiv preprint arXiv:2506.09991*, 2025b.
- Shinn Yao, Jeffrey Zhao, Shixiang Yu, and et al. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.
- Jiasheng Ye, Zaixiang Zheng, Yu Bao, Lihua Qian, and Quanquan Gu. Diffusion language models can perform many tasks with scaling and instruction-finetuning. *arXiv preprint arXiv:2308.12219*, 2023.
- Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 2025.
- Qiying Yu, Zheng Zhang, Ruofei Zhu, et al. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.
- Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*, 2025.
- Jason Zhu and Hongyu Li. Towards concise and adaptive thinking in large reasoning models: A survey, 2025. URL https://arxiv.org/abs/2507.09662.

# A. Appendix

#### A.1. Supervised Fine-Tuning (SFT) Details

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

Parameter	Value
Batch Size	1
<b>Gradient Accumulation Steps</b>	8
Learning Rate	$1 \times 10^{-5}$
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

Table 7 | SFT Training Configuration for DeepSeek-R1-Distill-Qwen-1.5B

Parameter	Value
Batch Size	1
<b>Gradient Accumulation Steps</b>	4
Learning Rate	$2 \times 10^{-5}$
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

Table 8 | SFT Training Configuration for DeepSeek-R1-Distill-Qwen-7B

# A.2. 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 below:

# 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 Re(z) < 0 and 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  $\theta$  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 25. </summary>

## A.3. Examples of parallel answers

We illustrate the effectiveness of ParaThinker through an example. Question: "Jen enters a lottery by picking 4 distinct numbers from  $S = \{1, 2, 3, \dots, 9, 10\}$ . 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." ParaThinker produces the following output (shortened for brevity):

# 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 116.

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