

000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 PARATHINKER: NATIVE PARALLEL THINKING AS A NEW PARADIGM TO SCALE LLM TEST-TIME COM- PUTE

006
007
008
009
010
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
032
033
034
035
036
037
038
039
040
041
042
043
044
045
046
047
048
049
050
051
052
053
Anonymous authors

Paper under double-blind review

ABSTRACT

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 that 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 thought 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 (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 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

054 that the model’s underlying capability is not the primary bottleneck; rather, the way we orchestrate
 055 test-time compute can be improved.

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

061 Based on these insights, we argue that the reasoning process of LLMs should be executed in a
 062 parallel, multi-threaded manner. By ensuring each thinking thread operates independently, we can
 063 mitigate Tunnel Vision and foster a diversity of thought. Furthermore, parallel thinking offers sig-
 064 nificant deployment advantages, as the decoding process can be batched to better utilize memory
 065 bandwidth, which in turn leads to improved arithmetic intensity (Williams et al., 2009) (the ratio of
 066 floating-point operations to total data movement).

067 To put parallel thinking into practice, we introduce an end-to-end solution, ParaThinker, which
 068 enables native parallel thinking in LLMs by allowing the model to generate diverse thoughts and
 069 aggregate them into a final answer. The major challenges to develop ParaThinker include how to in-
 070 duce thought diversity and how to avoid thought conflict, which we address by introducing trainable
 071 control tokens to trigger distinct reasoning trajectories, thought-specific positional embeddings to
 072 distinguish different paths, and a two-phase attention mask design that enforces independence dur-
 073 ing reasoning and controlled integration during summarization. Specifically, our solution features
 074 three core innovations:

- 075 • **Specialized Control Tokens:** We introduce a set of trainable tokens (e.g. `<think i>`) to ex-
 076 plicitly guide the model’s generation. Each `<think i>` token prompts the model to initiate a
 077 distinct reasoning path, which ensures diversity in reasoning.
- 078 • **Thought-Specific Positional Embedding:** To resolve positional ambiguity when merging par-
 079 allel thoughts, we augment the standard positional encoding with a unique, learnable embedding
 080 for each reasoning path. This allows the model to unambiguously differentiate the origin of each
 081 token during the final summarization stage.
- 082 • **SFT Training Pipeline:** We employ a scalable supervised fine-tuning (SFT) strategy where the
 083 model is trained on reasoning paths sampled from a teacher model. By randomly assigning the
 084 specialized `<think i>` tokens during this process, the model learns to generalize, enabling it to
 085 generate more parallel paths at inference time than were seen during training.

086 We evaluate ParaThinker on challenging math and coding benchmarks: AIME 2024, AIME 2025,
 087 AMC 2023, MATH-500 (Hendrycks et al., 2021), and LiveCodeBench v6 (Jain et al., 2024) against
 088 baselines such as standard autoregressive reasoning (DeepSeek-AI, 2025), majority voting (Chen
 089 et al., 2024a), and re-prefilling. Our approach demonstrates a remarkable leap in performance,
 090 achieving significantly improved accuracy with additional benefits of parallel computing. This ef-
 091 ficiency allows smaller LLMs equipped with our native thought parallelism to outperform much
 092 larger, standard reasoning models, charting a new path for scaling test-time compute.

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

100 2 UNDERSTANDING THE SCALING BOTTLENECK

101 2.1 IS THE BOTTLENECK DUE TO LLM CAPABILITY OR SCALING STRATEGY?

102 To empirically ground our approach, we first characterize the limitations of conventional test-time
 103 scaling. We start by evaluating the DeepSeek-R1-distill-Qwen-1.5B model (DeepSeek-AI, 2025)
 104 on the AIME 2024 benchmark under various computational budgets. We control the budget by im-

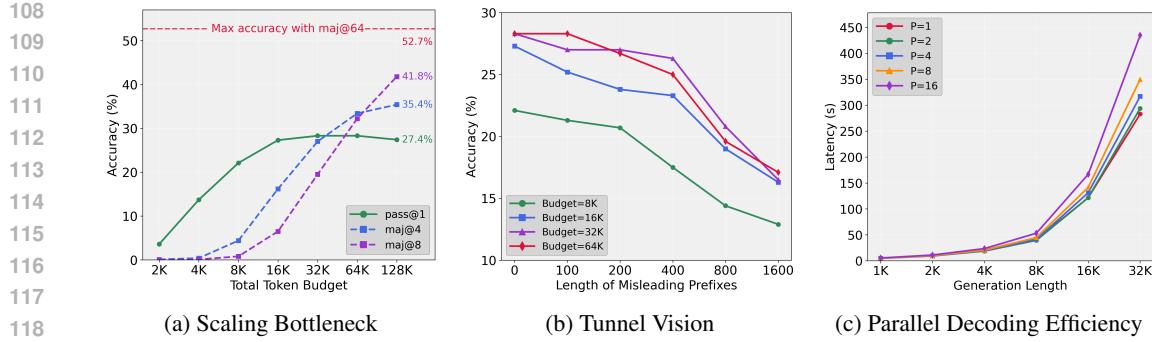


Figure 1: Diagnosing the limitations of sequential reasoning and the potential of parallelism. All experiments use DeepSeek-R1-Distill-Qwen-1.5B on the AIME 2024 benchmark. (a) Scaling Bottleneck: Accuracy against the total number of token budget (for majority voting *e.g.* maj@4, the total token budget 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 $n \in \{0, 100, \dots, 1600\}$, 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 $n \in \{1K, \dots, 32K\}$.

posing 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 ($</\text{think}>$). 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 the fundamental cause of this bottleneck to determine how to further bootstrap LLM test-time scaling. 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 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 1b, 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.

162

3 WHY NATIVE THOUGHT PARALLELISM?

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

169 However, majority voting is a heuristic—it applies the simple, fixed rule of choosing the most frequent
 170 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
 171 with easily quantifiable answers (e.g., multiple-choice or numeric values), failing on open-ended
 172 domains like complex agentic workflows, coding, or long-form proofs. **(2) Information loss:** Majority
 173 voting considers only the final answer, instead of the reasoning path, ignoring the valuable
 174 rationale, evidence, and intermediate steps within each reasoning path, making it impossible to syn-
 175 thesize insights from the full reasoning process to arrive at a better conclusion.

176 To overcome these limitations, we propose **native parallel reasoning** as an end-to-end solution. In
 177 this approach, we train the LLM to not only generate multiple reasoning paths in parallel but also to
 178 analyze on all the reasoning paths and generate the answer. Instead of relying on a fixed heuristic,
 179 the model learns to directly process and synthesize the full token trajectories of its parallel thoughts
 180 to produce a final, consolidated answer. This method makes the aggregation process itself trainable,
 181 allowing the model to learn optimal strategies for combining evidence, and path-aware, as it pre-
 182 serves the valuable information contained within each reasoning chain. We formalize *Why native*
 183 *parallel reasoning has the potential to outperform majority voting* theoretically in Appendix A.10.

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

190 Prior works on parallel computation typically relies on external verifiers for search (Snell et al., 2024;
 191 Ghosal et al., 2025), which introduce a scalability bottleneck. Several concurrent works (Zhao et al.,
 192 2025b; Yang et al., 2025b; Zheng et al., 2025; Fu et al., 2025) have also investigated the parallel rea-
 193 soning mechanism of LLMs, while they struggle to achieve substantial accuracy improvement due
 194 to positional ambiguity, computational inefficiency and/or limited thought diversity. More related
 195 works are discussed in Appendix A.2.

197

4 MODEL DESIGN

200 **Preliminaries of conventional sequential reasoning.** We denote an LLM by π_θ , where θ is the
 201 set of model parameters. Given an input prompt of l tokens $x = \{x_i\}_{i=1}^l$. The LLM then au-
 202 toregressively generates an output sequence $y = (y_1, y_2, \dots, y_L)$ with the conditional probability:
 203 $\pi_\theta(y|x) = \prod_{t=1}^L \pi_\theta(y_t|x, y_{<t})$. For tasks requiring multi-step reasoning, the output y can be de-
 204 composed into a reasoning path r followed by a final answer a : $y = (r, a)$. During decoding, each
 205 new token y_t requires attention over the full context $x, y_{<t}$, which involves computing Key (K) and
 206 Value (V) tensors. To avoid recomputation when generating each y_t , LLMs often use a KV-cache to
 207 store K/V tensors.

208

4.1 PARATHINKER WORKFLOW

209 As shown in Figure 2, our approach extends the sequential reasoning LLM paradigm by first gener-
 210 ating a set of P distinct reasoning paths $\{r^{(1)}, r^{(2)}, \dots, r^{(P)}\}$ for a single input x in parallel. Each
 211 individual reasoning path $r^{(i)}$ is a sequence of tokens representing a unique line of thought, sampled
 212 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
 213 that helps initiate a distinct reasoning path, which will be detailed in Section 4.2.

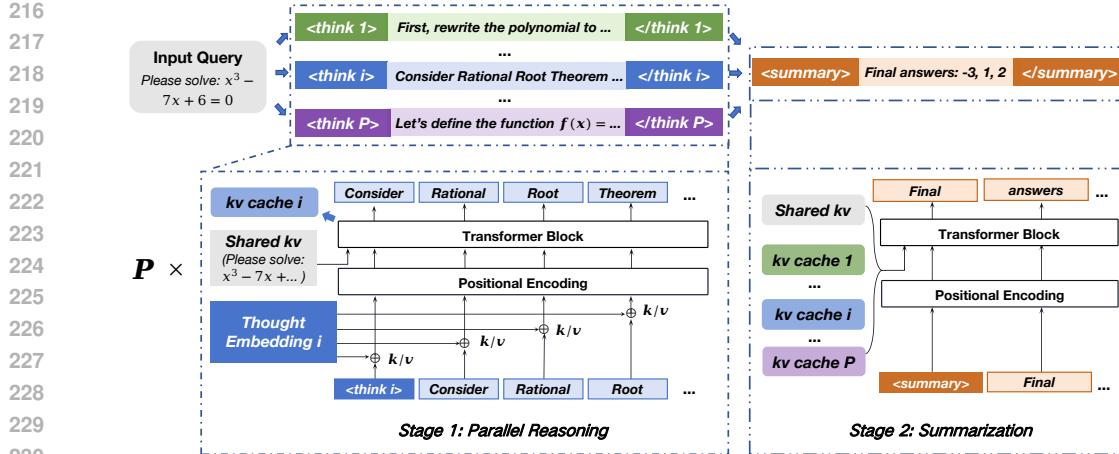


Figure 2: ParaThinker architecture. For an input question, ParaThinker processes it in two stages: (1) Parallel Reasoning: ParaThinker generates P reasoning paths in parallel; (2) Summarization: ParaThinker merges the reasoning paths by reusing their KV-caches to generate the final answer.

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

4.2 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: $\langle\text{think } i\rangle$, $\langle/\text{think } i\rangle$, $\langle\text{summary}\rangle$, and $\langle/\text{summary}\rangle$ for $i \in \{1, \dots, P\}$ to control the parallelization and merging operations. The $\langle\text{think } i\rangle$ 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 $\langle/\text{think } i\rangle$ token marks the end of a specific path, and the generation of the final answer is then wrapped within $\langle\text{summary}\rangle$ and $\langle/\text{summary}\rangle$ 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.

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.

270 **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:

$$277 \quad \tilde{k}_t^{(j)} = R_t(k_t^{(j)} + T^{(j)}) \quad \tilde{v}_t^{(j)} = v_t^{(j)} + T^{(j)} \quad (1)$$

279 Here, l_{max} denotes the maximum token number for each reasoning path, and R_t is the corresponding 280 RoPE rotation matrix. Using the RoPE property $(R_n)^T R_m = R_{m-n}$, the dot product attention score 281 between a query q_n from the summary (at local position n) and a key $\tilde{k}_m^{(j)}$ from path j (at position 282 m) is:

$$284 \quad \text{score}(n, m) = (R_n q_n^{(i)})^T \tilde{k}_m^{(j)} = (R_n q_n^{(i)})^T [R_m(k_m^{(j)} + T^{(j)})] = \underbrace{q_n^T R_{m-n} k_m^{(j)}}_{\text{Content-to-Content}} + \underbrace{q_n^T R_{m-n} T^{(j)}}_{\text{Content-to-Segment}} \quad (2)$$

288 The Content-to-Content term is the standard RoPE attention score, which calculates the relevance 289 between the query’s content (q_n) and the key’s content ($k_m^{(j)}$). This term is not related to the 290 reasoning path number j and thus does not change when scaling parallel reasoning paths. Content-to- 291 Segment term calculates the relevance between the query’s content (q_n) and the learnable identity of 292 the key’s entire reasoning path ($T^{(j)}$). This allows the query to directly probe for the origin of the 293 information. Because each reasoning path has a unique, trainable thought embedding, this term pro- 294 vides an unambiguous signal for the model to differentiate between parallel streams of text, solving 295 the positional ambiguity.

297 5 TRAINING AND DEPLOYMENT OF PARATHINKER

299 5.1 SCALABLE TRAINING DATA CURATION

301 ParaThinker models are trained by fine-tuning existing sequential reasoning LLMs with synthetic 302 parallel thought data. We design a scalable training data curation pipeline that consists of two key 303 components: multi-path training data scaling and extensible special tokens training.

304 **Multi-Path Training Data Scaling:** We develop a simple yet effective high-quality parallel reasoning 305 dataset by sampling multiple times from teacher reasoning LLM (*e.g.* DeepSeek-R1 (DeepSeek- 306 AI, 2025)). For a query x and groundtruth answer a , let the \hat{P} sampled answers denoted as 307 $\{(r^{(1)}, a^{(1)}), (r^{(2)}, a^{(2)}), \dots, (r^{(\hat{P})}, a^{(\hat{P})})\}$. We then concat each parallel answer into a groundtruth 308 answer with the format of: $\hat{y} = (<\text{think } 1>r^{(1)}</\text{think } 1>, \dots, <\text{think } \hat{P}>r^{(\hat{P})}</\text{think } \hat{P}> <\text{summary}>a</\text{summary}>)$. The resulting $Data_{r=1}^{\hat{P}} = (x, \hat{y})$ pairs are then used for SFT.

311 **Extrapolative Special Tokens Training:** Due to the high cost of teacher LLM inference, we 312 are often faced with the situation where we cannot generate enough reasoning paths when creating 313 $Data_{sft}$, *i.e.*, $\hat{P} < P$, where P denotes the maximum number of parallel reasoning paths supported 314 at inference time. Thus, during SFT stage, LLMs have to learn to extrapolate to $(r^{(\hat{P}+1)}, \dots, r^{(P)})$ 315 with training data $Data_{r=1}^{\hat{P}}$. We develop a dynamic special token sampling method for extrapolative 316 special tokens training. For each training batch, we randomly sample \hat{P} special tokens from 317 $<\text{think } i>, i \in \{1, \dots, P\}$ and prepend these sampled tokens to the beginning of each reasoning 318 sequence. Thus, each $<\text{think}_i>$ specializes to induce diverse inference-time trajectories, despite 319 training on only \hat{P} paths.

321 5.2 TRAINING AND INFERENCE IMPLEMENTATION

323 **Attention Mask Design.** ParaThinker employs a two-phase attention mask design to enable parallel 324 reasoning. In the reasoning phase, each path is decoded independently, with attention limited to

324 the input prompt and its own generated tokens, thereby preventing inter-path interference. In the
 325 summarization phase, answer tokens attend to the full prompt, all reasoning paths, and prior answer
 326 tokens, allowing integration across paths while preserving autoregressive consistency. More details
 327 are shown in Appendix A.3.

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

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

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

341 6 EXPERIMENTS

342 6.1 EXPERIMENTAL SETUP

343 **Training Details:** Our math reasoning experiments are based on a Qwen-2.5 (Qwen et al., 2025)
 344 1.5B and 7B model distilled from DeepSeek-R1 (DeepSeek-AI, 2025) (which we denote as original
 345 R1-1.5B and R1-7B below), and coding experiments are based on DeepSeek-R1-Distill-Qwen-1.5B.
 346 For math reasoning, we construct a parallel reasoning dataset with 6.2K problem-solution pairs, with
 347 each instance consisting of a query (x_i), ground-truth answer (a_i), and $\hat{P} = 6$ distinct reasoning
 348 paths. During every training step, we randomly choose a path number P from the set $\{2, 4, 6\}$
 349 and construct a training sample by concatenating P samples. For code reasoning, we construct a
 350 dataset of 50k problem-solution instances from OpenCodeReasoning (Ahmad et al., 2025), with
 351 questions that include more than 2 solutions. The final dataset includes either 2 or 4 alternative
 352 code-reasoning traces per problem ($\hat{P} \in \{2, 4\}$ during SFT). More details about training settings
 353 are listed in Appendix A.4.

354 **Baselines:** We compare ParaThinker against: (1) *Sequential*: Direct reasoning with original
 355 1.5B/7B models. (2) *Majority Voting*: Generate P independent paths and return the majority
 356 answer (Chen et al., 2024a). (3) *Re-Prefilling*: Generate P paths, concatenate them, and feed the full
 357 context into the model for summarization. This mimics ParaThinker’s summarization but is ineffi-
 358 cient since KV caches are not reused (Section 6.2). We do not include majority-voting for coding
 359 experiments since program correctness is often non-trivial to vote on.

360 **Benchmarks and Evaluation Setup:** We evaluate our model on 4 mathematical reasoning bench-
 361 marks: AIME 2024, AIME 2025, AMC 2023, and MATH-500 (Hendrycks et al., 2021), and 1
 362 coding benchmark: LiveCodeBench v6 (Jain et al., 2024). We use a token budget control method
 363 where each reasoning path is limited to a maximum of B tokens ($|r^{(i)}| \leq B$). If a model reaches
 364 this budget without naturally stopping, we enforce termination and then initiate the summarization
 365 stage by adding the (`<summary>`) token. This allows us to examine the utilization of the test-time
 366 scaling budget. More details of evaluation setup is shown in Appendix A.5.

367 6.2 SCALING PERFORMANCE

368 Table 1 and Table 2 compare ParaThinker with baseline methods under different token lengths.
 369 Compared with sequential LLMs, ParaThinker improves accuracy by up to 14.5% (1.5B) and 8.3%
 370 (7B) on AIME 2024, and by 3.6% (1.5B) and 8.8% (7B) on AIME 2025 at each token length on
 371 average (e.g. $2 \times 16K$ vs. $32K$), demonstrating the effectiveness of parallel reasoning. This indicates
 372 that the summarization stage captures a richer aggregation strategy than vote counting. For coding
 373 tasks, ParaThinker outperforms original LLM and data ablation LLM (trained on the same dataset
 374 with ParaThinker) with P increases, showing the potential to handle various tasks.

Method	Base model: R1-1.5B				Base model: R1-7B			
	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
Seq. (32K)	28.3	24.5	68.9	81.8	55.5	37.9	89.8	92.0
Seq. (64K)	27.1	25.5	67.7	81.7	56.0	39.6	89.8	92.5
Seq. (128K)	27.4	22.1	68.0	81.8	52.7	40.4	89.8	92.6
Maj. (2×16K)	25.9	23.0	67.0	81.4	52.3	38.3	88.4	91.4
Maj. (4×16K)	32.9	27.5	74.3	86.7	60.6	43.1	92.2	93.5
Maj. (8×16K)	41.0	31.8	79.8	89.0	68.8	49.6	93.1	94.2
Rep. (2×16K)	30.4	26.7	70.6	60.8	42.9	33.8	88.1	63.8
Rep. (4×16K)	24.2	25.8	61.3	58.6	43.3	33.3	86.3	63.2
Rep. (8×16K)	14.2	13.3	60.0	55.3	43.3	31.7	91.9	63.7
Para. (2×16K)	34.8	24.2	73.1	87.5	57.1	46.0	89.5	93.2
Para. (4×16K)	43.3	26.7	80.8	88.7	63.3	46.9	91.7	94.2
Para. (8×16K)	48.1	31.9	83.1	89.7	68.8	51.3	93.3	94.5

Table 1: Accuracy of ParaThinker and baselines (Seq., Maj., Rep. Para. denote Sequential, Majority Voting, Reprefill, and ParaThinker respectively). 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 .

Model	Original Sequential LLM				Fine-tuned Sequential LLM				ParaThinker-1.5B		
	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 ParaThinker, original sequential LLM, and the sequential LLM fine-tuned with the same data as ours.

We analyze how performance scales with the number of parallel paths in Figure 3 and 4, 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 relations between ParaThinker and majority voting in detail. The result is shown in Table 3. We find that ParaThinker does not conflict with the majority voting. Instead, it can be combined with majority voting to achieve higher accuracy. The highest accuracy of ParaThinker-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 5 shows the latency of ParaThinker 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. The efficiency of our method gives us the proof that we can achieve greater accuracy through parallel scaling within acceptable inference latency.

6.3 ABLATION STUDY

Train Data: We test how much the performance gain of ParaThinker attribute to the training data. Specifically, we use the same data of ParaThinker (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. ParaThinker, on the other hand, outperforms the fine-tuned variant across all budgets, confirming its effectiveness.

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

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

P	Strategy			Pass Rate		
	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; $Para.$: ParaThinker-1.5B

	A24	A25	AMC	MATH	Avg.
<i>R1-1.5B-SFT (Same dataset with ParaThinker)</i>					
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
<i>ParaThinker-1.5B</i>					
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 ParaThinker.

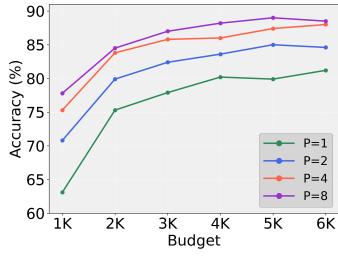


Figure 3: ParaThinker-1.5B Math-500 Scaling.

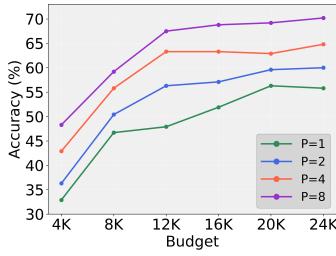


Figure 4: ParaThinker-7B AIME-24 Scaling.

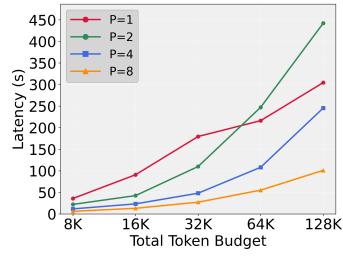


Figure 5: ParaThinker Latency under different budgets.

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 ParaThinker). We also report the $\text{pass}@P$ 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 ParaThinker 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 ParaThinker drops 0.6%-3.1%, 1.4%-4.3% respectively, demonstrating the necessity of these designs, where detailed analysis are stated in Appendix A.9.

7 CONCLUSION

Our work identifies a limitation of conventional sequential test-time compute scaling of LLMs named “Tunnel Vision”. We then introduce ParaThinker, a framework for native parallel reasoning that sidesteps the limitation by generating and aggregating multiple thought paths simultaneously. While our method presents a significant first step, future work could explore the interpretation of the learned aggregation strategy and end-to-end training with reinforcement learning.

486 8 REPRODUCIBILITY STATEMENT

488 For reproducibility, we provide detailed descriptions of our model architecture, training pipeline,
 489 and evaluation setup in Sections 4–6. Specifically, we describe the design of specialized control
 490 tokens, thought-specific positional embeddings, and the two-phase attention mask mechanism that
 491 enable native parallel reasoning. The training data curation process, including multi-path sampling
 492 from teacher models, extrapolative special token training, and dataset construction, is explained in
 493 Appendix A.4–A.6. Hyperparameters, optimization settings, and hardware configurations are also
 494 listed in Appendix A.4. All experiments are conducted with both 1.5B and 7B backbone models,
 495 and repeated across multiple reasoning budgets and path numbers to validate robustness. We further
 496 include ablation studies on positional embeddings, special tokens, termination strategies, and sum-
 497 marization methods to ensure the soundness of our conclusions. The implementation, datasets, and
 498 evaluation code will be released to the community to facilitate independent verification and further
 499 research.

500 9 ETHICS STATEMENT

503 Our study investigates methods to improve the reasoning efficiency and robustness of LLMs through
 504 parallel generation and aggregation of multiple reasoning paths. All benchmarks used in this work
 505 (AIME 2024, AIME 2025, AMC 2023, MATH-500, and LiveCodeBench v6) are publicly available,
 506 and no private or sensitive user data was involved. The proposed ParaThinker framework focuses
 507 on improving computational utilization and reasoning accuracy without altering semantic content in
 508 ways that could create additional risks of harmful or biased generation. While parallel reasoning
 509 may, in principle, be misused to amplify outputs, our design centers on controlled aggregation and
 510 transparent evaluation to mitigate such concerns. We adhere to the licensing requirements of all
 511 datasets and models used, and all data handling follows applicable privacy and legal standards. The
 512 authors declare no conflicts of interest or external sponsorship that could improperly influence this
 513 work. We encourage reviewer and community feedback on additional ethical considerations relevant
 514 to the deployment of parallel reasoning frameworks.

515 REFERENCES

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

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

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

528 Lingjiao Chen, Jared Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and
 529 James Zou. Are more llm calls all you need? towards the scaling properties of
 530 compound ai systems. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Pa-
 531 quet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Process-
 532 ing Systems*, volume 37, pp. 45767–45790. Curran Associates, Inc., 2024a. URL
 533 [https://proceedings.neurips.cc/paper_files/paper/2024/file/
 534 51173cf34c5faac9796a47dc2fdd3a71-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2024/file/51173cf34c5faac9796a47dc2fdd3a71-Paper-Conference.pdf).

535 Mouxiang Chen, Binyuan Hui, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Jianling Sun, Junyang Lin, and
 536 Zhongxin Liu. Parallel scaling law for language models, 2025a. URL <https://arxiv.org/abs/2505.10475>.

538 Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu,
 539 Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do not

540 think that much for $2+3=?$ on the overthinking of o1-like llms, 2025b. URL <https://arxiv.org/abs/2412.21187>.

541

542

543 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 <https://openreview.net/forum?id=4zAHgkiCQg>.

544

545

546 Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu
547 Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, Nicholas Thumiger, Aditya Desai, Ion
548 Stoica, Ana Klimovic, Graham Neubig, and Joseph E. Gonzalez. The danger of overthinking:
549 Examining the reasoning-action dilemma in agetic tasks, 2025. URL <https://arxiv.org/abs/2502.08235>.

550

551 DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,
552 2025. URL <https://arxiv.org/abs/2501.12948>.

553

554 Chenrui Fan, Ming Li, Lichao Sun, and Tianyi Zhou. Missing premise exacerbates overthinking:
555 Are reasoning models losing critical thinking skill?, 2025. URL <https://arxiv.org/abs/2504.06514>.

556

557 Guhao Feng, Yihan Geng, Jian Guan, Wei Wu, Liwei Wang, and Di He. Theoretical benefit and lim-
558 itation of diffusion language model, 2025. URL <https://arxiv.org/abs/2502.09622>.

559

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

562

563 Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson,
564 Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with
565 latent reasoning: A recurrent depth approach. *CoRR*, abs/2502.05171, February 2025. URL
566 <https://doi.org/10.48550/arXiv.2502.05171>.

567

568 Soumya Suvra Ghosal, Souradip Chakraborty, Avinash Reddy, Yifu Lu, Mengdi Wang, Dinesh
569 Manocha, Furong Huang, Mohammad Ghavamzadeh, and Amrit Singh Bedi. Does thinking
570 more always help? understanding test-time scaling in reasoning models, 2025. URL <https://arxiv.org/abs/2506.04210>.

571

572 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.

573

574

575 Xinyu Guan, Li Lyra Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang.
576 rstar-math: Small llms can master math reasoning with self-evolved deep thinking, 2025. URL
577 <https://arxiv.org/abs/2501.04519>.

578

579 Nathan Habib, Clémentine Fourrier, Hynek Kydlíček, Thomas Wolf, and Lewis Tunstall. Lighte-
580 eval: A lightweight framework for llm evaluation, 2023. URL <https://github.com/huggingface/lighteval>.

581

582 Zhengfu He, Tianxiang Sun, Qiong Tang, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. Dif-
583 fusionBERT: Improving generative masked language models with diffusion models. In Anna
584 Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meet-
585 ing of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4521–4534,
586 Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.
587 acl-long.248. URL <https://aclanthology.org/2023.acl-long.248/>.

588

589 Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian
590 Yu, Zhenwen Liang, Wenxuan Wang, et al. Deepmath-103k: A large-scale, challenging, de-
591 contaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint
592 arXiv:2504.11456*, 2025.

593

594 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song,
595 and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*,
596 2021.

594 Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL <https://github.com/huggingface/open-r1>.
 595
 596

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

600 Tian Jin, Ellie Y. Cheng, Zack Ankner, Nikunj Saunshi, Blake M. Elias, Amir Yazdanbakhsh,
 601 Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. Learning to keep a promise:
 602 Scaling language model decoding parallelism with learned asynchronous decoding, 2025. URL
 603 <https://arxiv.org/abs/2502.11517>.

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

608 Bespoke Labs. Bespoke-stratos: The unreasonable effectiveness of reasoning distilla-
 609 tion. www.bespokelabs.ai/blog/bespoke-stratos-the-unreasonable-effectiveness-of-reasoning-
 610 distillation, 2025. Accessed: 2025-01-22.

611
 612 Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative
 613 decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

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

616
 617 Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan
 618 Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step, 2023. URL
 619 <https://arxiv.org/abs/2305.20050>.

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

623
 624 OpenAI. Openai o1 system card. 2024. URL <https://arxiv.org/abs/2412.16720>.

625 OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL <https://arxiv.org/abs/2508.10925>.

626
 627 Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer,
 628 and Alane Suhr. Learning adaptive parallel reasoning with language models, 2025. URL <https://arxiv.org/abs/2504.15466>.

629
 630 Xiangyu Qi, Ashwinee Panda, Kaifeng Lyu, Xiao Ma, Subhrajit Roy, Ahmad Beirami, Prateek
 631 Mittal, and Peter Henderson. Safety alignment should be made more than just a few tokens
 632 deep. In *The Thirteenth International Conference on Learning Representations*, 2025. URL
 633 <https://openreview.net/forum?id=6Mxhg9PtDE>.

634
 635 Yuxiao Qu, Matthew Y. R. Yang, Amrit Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan
 636 Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement fine-
 637 tuning, 2025. URL <https://arxiv.org/abs/2503.07572>.

638
 639 Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan
 640 Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang,
 641 Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin
 642 Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li,
 643 Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang,
 644 Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.
 645 URL <https://arxiv.org/abs/2412.15115>.

646 Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazarian,
 647 Anton Sinitzin, Denis Kuznedelev, and Dan Alistarh. Hogwild! inference: Parallel llm
 generation via concurrent attention, 2025. URL <https://arxiv.org/abs/2504.06261>.

648 Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashishth Tiwari, Ruihang Lai, Jinyuan Shi, Ian
 649 Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput
 650 tradeoff for long context generation with speculative decoding. In Y. Yue, A. Garg, N. Peng,
 651 F. Sha, and R. Yu (eds.), *International Conference on Representation Learning*, volume 2025,
 652 pp. 6835–6850, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/13f972adf12bdf886583d48cd528002f-Paper-Conference.pdf.

653

654 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang,
 655 Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathe-
 656 matical reasoning in open language models, 2024. URL <https://arxiv.org/abs/2402.03300>.

657

658 Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally
 659 can be more effective than scaling model parameters, 2024. URL <https://arxiv.org/abs/2408.03314>.

660

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

663

664 Yi Sun, Han Wang, Jiaqiang Li, Jiacheng Liu, Xiangyu Li, Hao Wen, Yizhen Yuan, Huiwen Zheng,
 665 Yan Liang, Yuanchun Li, and Yunxin Liu. An empirical study of llm reasoning ability under strict
 666 output length constraint. *arXiv preprint arXiv:2504.14350*, April 2025. doi: 10.48550/arXiv.
 667 2504.14350.

668

669 Kimi Team et al. Kimi k1.5: Scaling reinforcement learning with llms, 2025. URL <https://arxiv.org/abs/2501.12599>.

670

671 Guanghai Wang, Yair Schiff, Subham Sahoo, and Volodymyr Kuleshov. Remasking discrete diffu-
 672 sion models with inference-time scaling. *arXiv preprint arXiv:2503.00307*, 2025.

673

674 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha
 675 Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language
 676 models. In *The Eleventh International Conference on Learning Representations*, 2023. URL
 677 <https://openreview.net/forum?id=1PL1NIMMrw>.

678

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

682

683 Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual per-
 684 formance 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>.

685

686 Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimiza-
 687 tion. In Jian Su, Kevin Duh, and Xavier Carreras (eds.), *Proceedings of the 2016 Confer-
 688 ence on Empirical Methods in Natural Language Processing*, pp. 1296–1306, Austin, Texas,
 689 November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1137. URL
 690 <https://aclanthology.org/D16-1137/>.

691

692 Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws:
 693 An empirical analysis of compute-optimal inference for problem-solving with language models,
 694 2025. URL <https://arxiv.org/abs/2408.00724>.

695

696 Haotian Xu, Xing Wu, Weinong Wang, Zhongzhi Li, Da Zheng, Boyuan Chen, Yi Hu, Shijia Kang,
 697 Jiaming Ji, Yingying Zhang, Zhijiang Guo, Yaodong Yang, Muhan Zhang, and Debing Zhang.
 698 Redstar: Does scaling long-cot data unlock better slow-reasoning systems?, 2025. URL <https://arxiv.org/abs/2501.11284>.

699

700 Rongwu Xu, Zehan Qi, and Wei Xu. Preemptive answer “attacks” on chain-of-thought reason-
 701 ing. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Associa-
 702 tion for Computational Linguistics: ACL 2024*, pp. 14708–14726, Bangkok, Thailand, August
 703 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.876. URL
 704 <https://aclanthology.org/2024.findings-acl.876/>.

702 An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jian-
 703 hong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical
 704 expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

705 An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025a. URL <https://arxiv.org/abs/2505.09388>.

708 Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. Multiverse: Your language mod-
 709 els secretly decide how to parallelize and merge generation. *arXiv preprint arXiv:2506.09991*,
 710 2025b.

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

714 Jiasheng Ye, Zaixiang Zheng, Yu Bao, Lihua Qian, and Quanquan Gu. Diffusion language models
 715 can perform many tasks with scaling and instruction-finetuning. *arXiv preprint arXiv:2308.12219*,
 716 2023.

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

720 Qiying Yu, Zheng Zhang, Ruofei Zhu, et al. Dapo: An open-source llm reinforcement learning
 721 system at scale, 2025. URL <https://arxiv.org/abs/2503.14476>.

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

725 Wenting Zhao, Pranjal Aggarwal, Swarnadeep Saha, Asli Celikyilmaz, Jason Weston, and Ilia Ku-
 726 likov. The majority is not always right: RL training for solution aggregation. *arXiv preprint
 727 arXiv:2509.06870*, 2025b.

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

732 Jason Zhu and Hongyu Li. Towards concise and adaptive thinking in large reasoning models: A
 733 survey, 2025. URL <https://arxiv.org/abs/2507.09662>.

735 **A APPENDIX**

737 **A.1 LLM USAGE STATEMENT**

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

741

742 **A.2 RELATED WORKS**

744 **A.2.1 SEQUENTIAL TEST-TIME SCALING**

746 Recent advances in test-time scaling seek to improve LLM reasoning by increasing computational
 747 depth during decoding, primarily through reinforcement learning (RL) (OpenAI, 2024; DeepSeek-
 748 AI, 2025; Google, 2025; Yang et al., 2025a; Team et al., 2025; Zhao et al., 2025a; Yu et al., 2025) and
 749 supervised fine-tuning (SFT) (Muennighoff et al., 2025; Ye et al., 2025). RL-based methods (Open-
 750 AI, 2024; Shao et al., 2024) encourage LLMs to allocate more computation to promising reasoning
 751 paths by encouraging self-reflection and iterative trial. Other approaches distill long-form rationales
 752 from larger teacher models into smaller student models, enabling deeper internal reasoning through
 753 fine-tuned CoT supervision (Labs, 2025; Ye et al., 2025; Xu et al., 2025; Geiping et al., 2025). While
 754 these methods significantly enhance LLM performance on complex tasks, they often suffer from in-
 755 creased 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
 755 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.

762 A.2.2 SEARCH-BASED METHODS FOR PARALLEL REASONING

764 Parallel Decoding LLMs improve reasoning by sampling multiple tokens at each step to accelerate
 765 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.,
 766 2023; Chen et al., 2023) and majority voting (Chen et al., 2024a). Recent advancements include
 767 Best-of-N (Lightman et al., 2023), Tree of Thoughts (ToT) (Yao et al., 2023), and Monte Carlo
 768 Tree Search (MCTS) (Snell et al., 2024; Guan et al., 2025). These approaches typically require
 769 an external verifier to evaluate and rank candidate completions, increasing computational cost and
 770 often relying on domain-specific or manually constructed reward signals. Our method departs from
 771 these paradigms by generating multiple reasoning trajectories internally and merging them using a
 772 lightweight summarization step, without requiring external verifiers or retraining.

774 A.2.3 NATIVELY PARALLEL GENERATION METHODS

776 Another line of work focuses on empowering LLMs to generate multiple tokens at each decoding
 777 iteration to accelerate LLMs theoretically. Diffusion-based language models (He et al., 2023; Ye
 778 et al., 2023; Zhao et al., 2025a; Wang et al., 2025; Arriola et al., 2025) sample multiple tokens in
 779 parallel during each diffusion step. While these methods can theoretically enable parallel generation,
 780 recent theoretical analyses (Feng et al., 2025) shows that for tasks involving sequential dependencies
 781 (e.g. reasoning), the number of required diffusion steps can scale linearly with sequence length,
 782 undermining their efficiency. PARSCALE (Chen et al., 2025a) investigates architectural parallelism
 783 by duplicating the input multiple times, applying distinct transformations, and aggregating outputs
 784 token-wise. However, this approach still requires architectural changes and specialized continual
 785 pretraining. In contrast, our approach retains the standard LLM architecture and introduces parallelism
 786 at the reasoning level by generating and caching multiple distinct chains of thought, which are
 787 later summarized into a final answer. Other works (Yang et al., 2025b; Pan et al., 2025; Rodionov
 788 et al., 2025; Jin et al., 2025) propose to automatically identify subtasks that can be solved in parallel.
 789 While effective for compositional tasks, it relies on explicit subtask decomposition, and these works
 790 focus on efficiency rather than accuracy improvement. ParaThinker, on the other hand, does not
 791 assume any subtask structure and improves both efficiency and accuracy by mitigating single-path
 792 failure cases (e.g. hallucinations or local optima) via diversity in reasoning. By integrating multiple
 793 KV caches in a summarization stage, our method scales inference without sacrificing correctness or
 794 requiring verifier models.

794 A.3 ATTENTION MASK DESIGN

796 To enable efficient parallel reasoning in existing LLM infrastructures during both training and in-
 797 ference, ParaThinker adopts a two-phase attention mask design. During the reasoning phase, each
 798 reasoning path is decoded independently, with attention restricted to the input prompt and its own
 799 token history. Let $M_{i,j}$ denote the attention mask between the index i and index j , where attention
 800 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
 801 path ($r^{(i)}$) is defined as:

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

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

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

810
811
812
813
814

$$M_{t,j}^A = \begin{cases} 0, & \text{if } j \leq t \text{ and } j \in \{1, \dots, l_x\} \cup \bigcup_{i=1}^P \text{Ind}_i \cup \text{Ind}_a \\ -\infty, & \text{otherwise} \end{cases} \quad (4)$$

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

817

A.4 TRAINING DETAILS

819
820
821

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

822
823
824
825
826
827

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.

828
829
830
831
832
833
834
835
836
837
838
839
840

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

841
842

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

843
844
845
846
847
848
849
850
851
852
853
854
855

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

856
857
858
859
860
861
862
863

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

864
865

A.5 EVALUATION SETTINGS

866

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 $\text{pass}@1 = \frac{1}{k} \sum_{i=0}^k p_i$, 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.

867

877

A.6 EXAMPLES OF TRAINING DATA

878

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:

879

880

881

882

883

884

885

886

SFT Data Example

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

<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 25.</summary>

918 A.7 EXAMPLES OF PARALLEL ANSWERS
919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

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):

972
973

Answer Example

974

<think1>

975

Use the conditional probability formula

976

977

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

978

979

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

980

981

982

983

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

984

985

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

986

987

988

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

989

The final conditional probability is

990

991

992

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

993

hence $m + n = 116$.

994

</think1>

995

996

<think2>

997

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

998

999

1000

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

1001

1002

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

1003

<think3>

1004

Directly compute the intersection to confirm:

1005

1006

1007

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

1008

1009

<think4>

1010

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

1011

1012

1013

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

1014

1015

1016

<summary>

1017

By analyzing multiple reasoning processes above, I concluded that: The final answer is

1018

116.

1019

</summary>

1020

1021

1022

1023

1024

1025

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.

		$P=2$	$P=4$	$P=8$
1026	Last-Finish	32.1	37.1	42.5
1027	Half-Finish	34.8	38.3	42.5
1028	First-Finish (Default)	34.8	43.3	48.1
1029				
1030				

Table 8: 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.

		$P=1$	$P=2$	$P=4$	$P=8$
1035	<i>pass@1</i>	51.9	57.1	63.3	68.8
1036	<i>maj@4</i>	60.6	64.9	70.3	74.7
1037	<i>maj@8</i>	68.8	68.9	72.1	77.6
1038	<i>maj@16</i>	73.3	70.0	73.3	76.7
1039					
1040					

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

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 TERMINATION STRATEGY ABLATION

We ablate the special control tokens (*e.g.* `<think i>`) and thought-specific positional embeddings in ParaThinker 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 ParaThinker 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

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 6 shows the performance degradation as P increases on AIME 2024. As illustrated, the Flatten-PE ParaThinker 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.

	$P=2$	$P=4$	$P=8$
ParaThinker-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, $t = 0.5$, $B = 16K$).

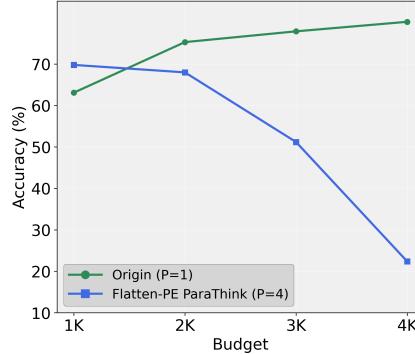


Figure 6: **Flattened PE Ablation:** Comparison on R1-1.5B trained with Flattened PE and original model (sequential) under Math-500 dataset.

A.10 WHY PARATHINKER OUTPERFORMS MAJORITY VOTING

Setup and notation. Let $(q, R, a) \sim \mathcal{D}$, where q is the input query, $R = (r^{(1)}, \dots, r^{(P)})$ 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 $p_{\text{data}}(a | q, R)$. Let $A = f(R) = (a^{(1)}, \dots, a^{(P)})$ be the per-path answers extracted deterministically from R . An *aggregator* is any conditional distribution $p(a | q, R)$; we focus on two classes:

- **Voting-only aggregators:** $p(a | q, R)$ that depend on R only through A , i.e. $p(a | q, R) = p(a | q, A)$.
- **Path-aware aggregators:** $p(a | q, R)$ 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,

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

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

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

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

$$I(a; R | A, q) = \mathbb{E}_{(q, A)} \mathbb{E}_{R|q, A} \left[D_{\text{KL}}(p_{\text{data}}(\cdot | q, R) \| p_{\text{data}}(\cdot | q, A)) \right].$$

1134 Equivalently (taken as expectation over (q, R)),
 1135

$$1136 \quad I(a; R | q) - I(a; A | q) = \mathbb{E}_{(q, R)} \left[D_{\text{KL}}(p_{\text{data}}(\cdot | q, R) \| p_{\text{data}}(\cdot | q, A)) \right].$$

1138 We will call the left-hand side the *information gap*.
 1139

1140 **Optimal KL for voting-only aggregators.** For any voting-only aggregator $p(a | q, A)$, the ex-
 1141 pected forward KL to the data posterior is

$$1142 \quad \mathbb{E}_{(q, R)} \left[D_{\text{KL}}(p_{\text{data}}(\cdot | q, R) \| p(\cdot | q, A)) \right].$$

1144 For each (q, A) , the choice of $p(\cdot | q, A)$ minimizing the inner expectation over $R | q, A$ is precisely
 1145 $p_{\text{data}}(\cdot | q, A)$. Therefore the minimal achievable expected KL over all voting-only aggregators
 1146 equals the information gap:

$$\begin{aligned} 1147 \quad & \inf_{p(\cdot | q, A)} \mathbb{E}_{(q, R)} \left[D_{\text{KL}}(p_{\text{data}}(\cdot | q, R) \| p(\cdot | q, A)) \right] \\ 1149 \quad & = \mathbb{E}_{(q, R)} \left[D_{\text{KL}}(p_{\text{data}}(\cdot | q, R) \| p_{\text{data}}(\cdot | q, A)) \right] \\ 1151 \quad & = I(a; R | q) - I(a; A | q). \end{aligned}$$

1153 Thus the information gap is an exact lower bound on the expected KL that any voting-only aggre-
 1154 gator must incur. In fact, this bound is only attained by the Bayes-optimal choice $p(a | q, A) =$
 1155 $p_{\text{data}}(a | q, A)$, which requires full knowledge of the data distribution. Heuristic rules such as
 1156 majority voting are far more restrictive and generally achieve strictly larger expected KL.

1157 **ParaThinker (KL minimizer) vs. voting.** Let $\mathcal{F} = \{p_\theta(a | q, R) : \theta \in \Theta\}$ be a family of
 1158 path-aware aggregators (e.g., parameterized models trained with SFT on full paths). Define
 1159

$$1160 \quad \theta^* = \arg \min_{\theta \in \Theta} \mathbb{E}_{(q, R)} \left[D_{\text{KL}}(p_{\text{data}}(\cdot | q, R) \| p_\theta(\cdot | q, R)) \right].$$

1162 Two observations follow immediately:

- 1163 1. If \mathcal{F} contains the Bayes-optimal voting aggregator $p_{\text{data}}(\cdot | q, A)$ as a special case (i.e.,
 1164 some $p_\theta(\cdot | q, R) = p_{\text{data}}(\cdot | q, A)$ for all (q, R)), then by minimality of θ^* ,
 1165 $\mathbb{E}[D_{\text{KL}}(p_{\text{data}} \| p_{\theta^*})] \leq \inf_{p(\cdot | q, A)} \mathbb{E}[D_{\text{KL}}(p_{\text{data}} \| p(\cdot | q, A))] = I(a; R | q) - I(a; A | q)$.
 1166
- 1168 2. If \mathcal{F} is sufficiently expressive to approximate $p_{\text{data}}(\cdot | q, R)$ well, then the left-hand side
 1169 above can be made small (approaching zero), whereas the right-hand side equals the infor-
 1170 mation gap and is strictly positive whenever $I(a; R | A, q) > 0$.

1171 Consequently, in any non-degenerate situation where intermediate reasoning tokens in R carry in-
 1172 formation about a beyond A , path-aware SFT (ParaThinker) can attain strictly lower expected KL
 1173 than any voting-only aggregator.

1174 **From KL to 0-1 risk.** Define the expected classification error (0-1 risk) of an aggregator p by

$$1176 \quad \mathcal{R}(p) := \Pr_{(q, R, a) \sim \mathcal{D}} [\hat{a} \sim p(\cdot | q, R) \text{ s.t. } \hat{a} \neq a].$$

1178 Pinsker's inequality implies, for each (q, R) ,

$$1180 \quad \text{TV}(p_{\text{data}}(\cdot | q, R), p_{\theta^*}(\cdot | q, R)) \leq \sqrt{\frac{1}{2} D_{\text{KL}}(p_{\text{data}}(\cdot | q, R) \| p_{\theta^*}(\cdot | q, R))}.$$

1181 Using the fact that the per-context increase in 0-1 risk is bounded by this TV distance, and taking
 1182 expectation, we obtain

$$1184 \quad \mathcal{R}(p_{\theta^*}) \leq \mathcal{R}(p_{\text{data}}) + \sqrt{\frac{1}{2} \mathbb{E}_{(q, R)} \left[D_{\text{KL}}(p_{\text{data}}(\cdot | q, R) \| p_{\theta^*}(\cdot | q, R)) \right]}.$$

1186 Combining this with the KL comparison above gives the desired qualitative statement: when the
 1187 information gap is positive, a path-aware aggregator that closely fits the data posterior will achieve
 1188 lower 0-1 risk than any voting-only aggregator.

1188
1189**Remarks and caveats.**1190
1191
1192

- 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 $I(X; Y | Z) = \mathbb{E}_Z \mathbb{E}_{Y|Z} D_{\text{KL}}(P_{X|Y,Z} \| P_{X|Z})$.
- 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 θ^* may not reach the theoretical minimum. Nonetheless, the direction of the inequality indicates when and why Path-aware SFT (ParaThinker) 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.

1200

1201

1202

1203

1204

1205

1206

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

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241