

Branch Later, Compute Less, Think Better: Adaptive Depth Parallelism for Efficient Parallel Prefix-Tuning

Mingqian Zhang^{1,2}, Mouxiang Chen^{1,3}, Zhongxin Liu^{1,2,*}

¹College of Computer Science and Technology, Zhejiang University

²The State Key Laboratory of Blockchain and Data Security, Zhejiang University

³Qwen, Alibaba

{mingqianzhang, chenmx, liu_zx}@zju.edu.cn

Abstract

Model-internal parallelization improves generation quality by introducing multiple parallel computation paths, but its additional latency and computational overhead hinder practical deployment. Recent studies reveal a functional division across model depth: shallow layers primarily encode input-conditioned features, whereas deeper layers are more involved in reasoning and generation. Since parallel paths process the same input and are optimized toward the same generation objective, their early representations are likely to overlap substantially, suggesting that early-layer parallelism may induce redundant computation. Motivated by this observation, we propose **Adaptive Depth Parallelism (ADP)**, a framework that shares computation in shallow layers and activates parallel branches only after an adaptively selected bifurcation layer. We provide a sufficient-condition analysis for delayed branching and identify an empirical transition in inter-path cosine similarity across depth. This motivates using the stable minimum-similarity layer as a practical estimate of the bifurcation point. Experiments show that ADP achieves 104.83% of baseline performance while using only 39.4% of the computational load.

1 Introduction

Large language models (LLMs) have advanced natural language processing, yet their capabilities remain strongly tied to model scale (Hoffmann et al., 2022). Recent parallel frameworks, including ParScale (Chen et al., 2025), ParaThinker (Wen et al., 2025), and Multiplex Thinking (Tang et al., 2026), improve smaller models by introducing multiple computational paths. However, full-stack parallelization replicates computation across all Transformer layers, causing the computational cost of a P -path model to scale substantially beyond that of a single-path model. This additional cost over

single-path inference leads to higher latency and resource consumption, making such parallel models difficult to deploy in real-world scenarios (Alman and Song, 2024; Liao et al., 2025; Han et al., 2025). This motivates a central question: **Can we retain the benefits of full-stack parallelism while reducing the additional compute and memory overhead it introduces?**

A common design in parallel frameworks is to parallelize all layers uniformly, implicitly treating them as equally suitable for multi-path computation. Yet Transformer layers are functionally heterogeneous across depth (Tan et al., 2026; Fan et al., 2025c; Song et al., 2026): shallow layers mainly extract input-conditioned features and align semantics, whereas deeper layers are more involved in reasoning and final token prediction. This heterogeneity suggests that not every layer benefits equally from parallelism across paths.

Prior work suggests that representations associated with the same target tend to be similar (Hu et al., 2024; Huh et al., 2024; Ju et al., 2026). In parallel training, paths are optimized toward the same loss objective; during inference, they condition on the same generation goal while differing mainly through path-specific prefixes. This suggests that the shallow layers contain a large shared component, and the differences mainly reflect the perturbations caused by the prefix.

To make this view operational, ADP represents early paths as shared features plus compact path-specific perturbations. Rather than executing every shallow Transformer layer independently, ADP uses trainable path-specific prefixes after a shared shallow backbone to approximate these perturbations. With sufficient prefix capacity, such prefixes can compensate for early prefix-induced deviations that are later absorbed by deeper layers.

ParScale (Chen et al., 2025) has demonstrated that diversity is a source of performance for parallel frameworks. Therefore, we use cosine similarity to

*Corresponding author.

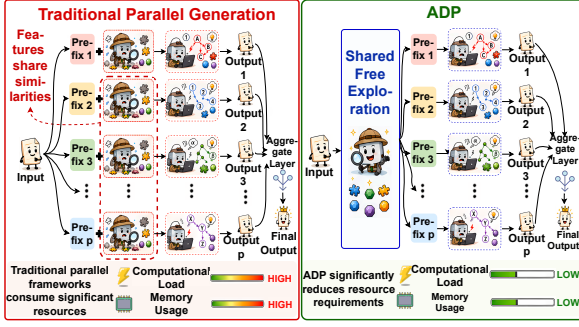


Figure 1: Comparison between full-stack parallelization and ADP. Conventional frameworks apply multiple paths to all layers, causing redundant shallow-layer feature extraction, whereas ADP shares early computation and parallelizes only later layers.

identify the transition from shared feature construction to path-specific inference. During shallow *free exploration*, prefix-induced perturbations persist, so inter-path similarity decreases slowly. In deeper *inference* layers, paths realign toward the common generation objective, producing a sharp similarity increase. Therefore, cosine similarity serves as a transition diagnostic, not as a criterion that lower similarity is always preferable. As shown in Figure 4, the stable minimum of the similarity curve provides a practical estimate of the transition point, supporting the use of trainable prefixes after a shared shallow backbone.

Based on these observations, we propose **Adaptive Depth Parallelism (ADP)**, a parallel training and inference framework aligned with the layer-wise functional hierarchy of LLMs. ADP uses a single-path backbone in early layers and expands into multiple branches only after an adaptively selected bifurcation layer. This design shares redundant early computation, reintroduces path-specific variation through trainable prefixes, preserves path-specific reasoning in later layers, and substantially reduces training and generation costs relative to full-stack parallelization.

To locate the bifurcation layer, ADP runs a short warm-up with a full stack framework, computes inter-path cosine similarity across layers, and selects the stable minimum as the start of parallelization. The warm-up is used only to estimate the transition point for a given model-path configuration and is amortized by the savings over full-stack parallel training. This practical estimator is supported by the observed similarity trend and our sufficient-condition analysis, and remains consistent across the evaluated model scales and tasks.

We evaluate ADP on code, mathematics, and choice questions benchmarks using models of different sizes. Compared with the full-stack parallel baseline under the same number of paths, ADP achieves 104.83% normalized performance with only 39.4% of the full-stack parallel computational load and 47.5% of the memory. In a larger-path setting, ADP achieves 100.39% normalized performance with only 26.6% of the full-stack parallel computational load and 31.7% of the memory.

Our contributions are summarized as follows:

1. We provide an analysis explaining when delayed parallelization can preserve performance under contraction assumptions, with a bounded loss relative to full-stack parallelization.
2. We propose ADP, a parallel inference framework that identifies a bifurcation layer from the cosine-similarity curve and expands a single path into multiple branches only after this layer, substantially reducing computation relative to full-stack parallelization during training and generation.
3. We validate ADP through extensive experiments and release code at [github](#).

2 Related Work

2.1 Foundations of LLMs and Scaling Laws

Large language models (LLMs) built on the Transformer architecture (Vaswani et al., 2017) have made substantial progress in recent years. Models such as GPT (OpenAI et al., 2024), Qwen (Yang et al., 2025), and DeepSeek (DeepSeek-AI et al., 2025) now achieve strong performance on challenging tasks. These gains, however, have largely coincided with substantial increases in model scale. Kaplan et al. (Kaplan et al., 2020) identified power-law relationships between language modeling loss and parameter count, dataset size, and compute. The Chinchilla scaling laws (Hoffmann et al., 2022) further refined this framework by emphasizing compute-optimal trade-offs between model size and training data. This trend motivates efficient frameworks that preserve performance without further increasing model scale.

2.2 Parallel Frameworks

Recent work has explored parallelization to improve the performance of smaller models by introducing multiple computational paths. ParaThinker (Wen et al., 2025) constructs multiple “chains of thought,” while Multiplex Thinking (Tang et al., 2026) introduces “multiplex tokens”

to enrich information representation. These methods enable parallel inference only; the model itself remains monolithic. ParScale (Chen et al., 2025), however, stands apart: it introduces—for the first time—a framework for parallel models, facilitating the training of natively parallel models during both the training and inference phases. ParScale further shows that diversity is a key mechanism by which parallel frameworks improve performance. However, the computational overhead of these methods limits their deployment in high-concurrency settings, motivating lower-cost parallel frameworks.

2.3 Internal Hierarchy of LLMs

LLMs do not process information uniformly across layers; instead, they exhibit a functional hierarchy (Huang et al., 2026). Song et al. (Song et al., 2026) use layer pruning to characterize the functions of different layers. Fan et al. (Fan et al., 2025c) show that deeper layers contribute more to reasoning. Tan et al. (Tan et al., 2026) examine layer-wise division of labor across model families. Collectively, these studies suggest that reasoning is concentrated in deeper layers. This observation motivates parallelizing only the layers that strongly affect diversity, rather than the entire network.

3 Can We Parallelize Only Deeper Layers?

Prior studies (Tan et al., 2026; Fan et al., 2025c; Song et al., 2026) suggest that shallow layers support free exploration, whereas deeper layers are more inference-oriented. We therefore test whether parallelism can begin only at a bifurcation layer l^* : preceding layers use a single path, and the hidden state at l^* is copied to P prefix-conditioned paths for subsequent parallel computation.

Using the 28-layer Qwen3-1.7B-Base, we set fixed bifurcations at layers 14 and 27. Layer 14 tests whether covering the inference-oriented layers preserves performance, whereas layer 27 represents an extreme delayed setting. In all cases, the base model is frozen, and only the P prefixes are trained. Figure 2 reports performance and cost.

We evaluate code generation, a reasoning-intensive setting (Li et al., 2023; Zheng et al., 2025; Jimenez et al., 2024; Fan et al., 2026). Figure 2 shows that layer-14 bifurcation maintains performance under the same parallel number, whereas layer-27 bifurcation degrades markedly. Thus, parallelism is most beneficial when it covers the infer-

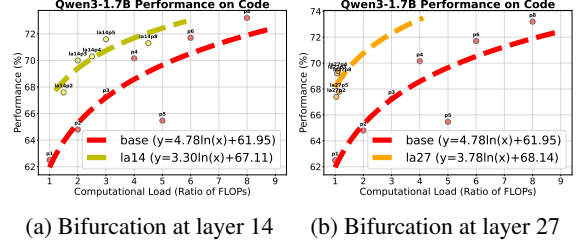


Figure 2: Fixed-layer delayed parallelization. ‘la’ denotes layer and ‘p’ number of paths; **Red** denotes Full, **Yellow** denotes Layer14, and **Orange** denotes Layer27.

ence layers; trainable prefixes can approximate earlier free-exploration perturbations, reducing computation while preserving performance.

4 Methods

Section 3 shows that delayed parallelism is feasible, but a fixed bifurcation layer may not generalize. Adaptive Depth Parallelism (ADP) therefore identifies this layer automatically and applies parallelism only afterward. Figure 3 overviews the method.

4.1 Analysis of the Conventional Framework

The parallel scaling law (Chen et al., 2025) attributes gains to inter-path diversity. We quantify this diversity with average cosine similarity and analyze its layer-wise dynamics.

Proposition 1. *Cosine similarity is controlled by the prefix-dependent directional updates $\eta_l^{(i)}$ and the existing directional deviations $\delta_l^{(i)}$. Here, $\bar{C}_l$ is the average pairwise cosine similarity among different paths at layer l , p is the number of paths, and i indexes a path. The term $\delta_l^{(i)}$ denotes the existing directional deviation of path i at layer l , whereas $\eta_l^{(i)}$ denotes the prefix-dependent directional update from layer l to layer $l + 1$.*

$$\bar{C}_{l+1} - \bar{C}_l = -\frac{1}{p-1} \left(2 \sum_{i=1}^p \langle \delta_l^{(i)}, \eta_l^{(i)} \rangle + \sum_{i=1}^p \|\eta_l^{(i)}\|^2 \right). \quad (1)$$

The proof is in Appendix B. Equation 1 shows that similarity decreases when prefix-dependent updates amplify path deviations and increases when later updates suppress them. Since paths sharing the same objective tend to learn similar representations (Hu et al., 2024; Huh et al., 2024; Ju et al., 2026), the observed divergence mainly reflects prefix-induced perturbations rather than se-

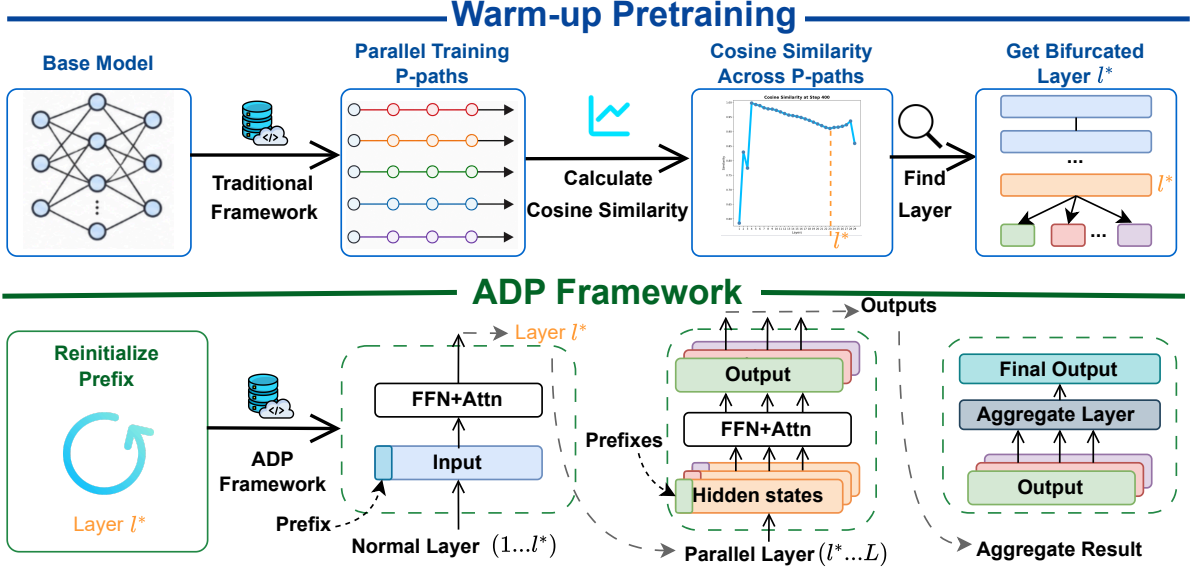


Figure 3: Overview of ADP.

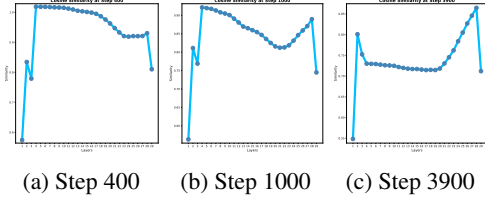


Figure 4: Evolution of inter-path cosine similarity during training.

mantic disagreement; these perturbations can be approximated by training new prefixes.

4.2 Locate Bifurcated Layer

Figure 4 shows a stable layer-wise turning point for fixed P : cosine similarity decreases before l^* and increases afterward. Appendix F.1 further shows that the minimum layer remains largely stable across training steps. Appendix A also shows that, after parallel execution begins, ADP hidden states progressively approach those of the traditional framework, suggesting that deeper layers suppress prefix-induced perturbations while preserving task-relevant semantics.

We therefore summarize this experimental finding as the following local hypothesis.

Assumption 1. *Post-bifurcation local contraction. For every $l \in \{l^*, \dots, L-1\}$ and path i , the normalized transition map is contractive on the local region containing the ADP and traditional-reference trajectories; that is, there exists $0 \leq \rho_l < 1$ such that*

$$\|\Phi_l^{(i)}(u) - \Phi_l^{(i)}(v)\| \leq \rho_l \|u - v\|, \quad u, v \in \mathcal{U}_l,$$

where $\Phi_l^{(i)}$ is the normalized transition of path i , and $\mathcal{U}_l$ is the region visited by the ADP trajectory and the traditional-framework reference trajectory. For normalized hidden state q and candidate bifurcation layer s , define the directional error between ADP and the traditional framework as

$$E_l^q(s) = \max_i \|q_l^{\text{ADP},(i)}(s) - q_l^{\text{full},(i)}\|.$$

Proposition 2. *Under Assumption 1, the directional error between ADP and the traditional framework contracts after l^* :*

$$E_L^q(l^*) \leq E_{l^*}^q(l^*) \prod_{k=l^*}^{L-1} \rho_k. \quad (2)$$

Proof. For any path i and layer $l \geq l^*$, ADP and the traditional reference use the same path-specific transition map:

$$\begin{aligned} q_{l+1}^{\text{ADP},(i)}(l^*) &= \Phi_l^{(i)}(q_l^{\text{ADP},(i)}(l^*)), \\ q_{l+1}^{\text{full},(i)} &= \Phi_l^{(i)}(q_l^{\text{full},(i)}). \end{aligned}$$

By Assumption 1,

$$\begin{aligned} &\|q_{l+1}^{\text{ADP},(i)}(l^*) - q_{l+1}^{\text{full},(i)}\| \leq \\ &\rho_l \|q_l^{\text{ADP},(i)}(l^*) - q_l^{\text{full},(i)}\|. \end{aligned}$$

Taking the maximum over paths gives

$$E_{l+1}^q(l^*) \leq \rho_l E_l^q(l^*). \quad (3)$$

Iterating from $l = l^*$ to $L - 1$ gives Equation 2. Since each $\rho_l < 1$, the discrepancy between ADP and traditional-framework hidden states is attenuated layer by layer after parallel execution starts. Equation 2 also explains why l^* is the optimal start layer. Starting from $l^* + 1$ misses the contraction at layer l^* and thus reduces the attainable discrepancy reduction. Starting from $l^* - 1$ occurs before the turning point, where states still expand, so early parallel execution may enlarge the discrepancy and provide little benefit. Therefore, l^* is the earliest point that avoids expansion while preserving the full post-transition contraction.

ADP locates this layer during warm-up. Specifically, it first trains for several hundred steps under the traditional framework, computes layer-wise average cosine similarity, and selects the stable minimum under a fixed parallel configuration P as the bifurcated layer:

$$l^* = \arg \min_{l \in [2, L-1]} \bar{S}^{(l)}. \quad (4)$$

Appendix E summarizes ADP.

We further demonstrate the stability of our method under different random seeds. Across three random seeds, Qwen3-1.7B with $P=2$ consistently selects Layer 17 within 150 steps and remains stable thereafter (Appendix F.2).

4.3 Adaptive Parallelization

After locating l^* , ADP runs the shared prefix from layer 1 to layer l^* and uses the resulting hidden state as the common bifurcation point. From this point, P parallel paths are generated and propagated through the remaining layers. Following ParScale (Chen et al., 2025), their outputs are fused by a lightweight aggregator with two linear layers and a SiLU activation. The trainable prefix-generating MLP learns to approximate the perturbations that full parallelism would otherwise introduce (Li and Liang, 2021). Thus, ADP preserves a shared early representation and applies parallel decoding only after hidden states begin to contract.

This bifurcated structure directly reduces computational cost. Full parallelism repeats all layers for all P paths, with cost

$$\text{FLOPs}_{\text{full}} = P \times \text{Cost}(1, L)$$

In contrast, ADP computes the prefix before l^* once and duplicates only the suffix, yielding

$$\begin{aligned} \text{FLOPs}_{\text{ADP}} &= \text{Cost}(1, l^*) \\ &\quad + P \times \text{Cost}(l^*, L). \end{aligned}$$

The first term is the shared computation before bifurcation, and the second is the parallel computation after bifurcation. Savings come from avoiding repeated execution of early layers, where parallel paths have not yet entered the contraction regime. Including the warm-up procedure for locating l^* , the training cost becomes

$$\begin{aligned} \text{FLOPs}_{\text{ADP}} &= \text{Cost}(1, l^*) + \frac{T}{N} \times P \\ &\quad \times \text{Cost}(1, L) + P \times \text{Cost}(l^*, L), \end{aligned}$$

where L is the number of layers, P is the number of paths, N is the total number of training steps, and T is the number of warm-up steps used to estimate l^* . Since $T \ll N$, $\frac{T}{N}$ is small, and the increased computation in stage warm-up is relatively small compared to the computational savings in training.

The computational reduction does not imply uncontrolled deviation from full parallelism. Because l^* is selected at the start of the contraction region, the discrepancy introduced after bifurcation remains bounded through subsequent layers. This yields the following loss-gap bound.

Proposition 3. *Let W denote the final linear head. The loss gap between ADP and full parallelism is bounded by*

$$|\mathcal{L}_{\text{ADP}} - \mathcal{L}_{\text{full}}| \leq \sqrt{2} \|W\|_2 E_L^q(l^*). \quad (5)$$

This inequality shows that the ADP–full-parallel loss gap is controlled by the spectral norm of the final linear head W and the final hidden-state discrepancy $E_L^q(l^*)$ induced by bifurcation at l^* . The factor $\|W\|_2$ measures how strongly the output head can amplify hidden-state differences, whereas $E_L^q(l^*)$ measures the discrepancy remaining after the post-bifurcation layers. Together with Proposition 2, this shows that choosing l^* at the onset of contraction reduces computation while keeping the final loss gap explicitly upper-bounded.

5 Evaluation

We train models at several scales and evaluate ADP across code, mathematics, and choice tasks.

5.1 Experiment Setup

We organize the evaluation around three research questions:

RQ1: Effectiveness: Does ADP reduce computational load, and does it outperform traditional parallel frameworks under the same load?

RQ2: Ablation: Does the adaptive layer-selection module outperform fixed bifurcation layers at comparable computational cost?

RQ3: Layer Analysis: Are the layers selected by ADP reasoning layers identified in prior work?

RQ4: Memory Analysis: Can ADP reduce GPU memory usage?

RQ5: Wall-clock Efficiency: Does ADP improve efficiency?

5.2 Datasets and Base Models

We evaluate ADP on standard benchmarks for code generation, mathematical reasoning, and choice questions, which evaluate the model’s reasoning and generative capabilities (Fan et al., 2025a). Because model scale affects task difficulty—smaller models struggle with complex tasks, whereas larger models saturate on simpler ones—we follow Yang et al. (Yang et al., 2025) and use scale-specific evaluation protocols.

We train Qwen3-0.6B-Base, Qwen3-1.7B-Base, and Qwen3-4B-Base. We pre-train 0.8B tokens from StackV2-Python (Lozhkov et al., 2024) and then perform supervised fine-tuning on 0.2B tokens from Magicoder (Wei et al., 2024). Detailed training settings are provided in Appendix D.

We select ParScale (Chen et al., 2025), which achieves parallelism within the internal structure of the model, as our traditional framework. ADP’s aggregate layer and prefix parameters align with ParScale to ensure fairness.

Code: Code generation is sensitive to fine-grained implementation decisions, where small deviations in critical code segments can determine functional correctness (Fan et al., 2025b). For Qwen3-0.6B-Base and Qwen3-1.7B-Base, we evaluate on HumanEval and MBPP using EvalPlus (Liu et al., 2023). Because smaller models are unstable under greedy decoding, we also evaluate them with temperature sampling at a temperature of 0.7. We also test the performance of greedy decoding.

For Qwen3-4B-Base, we use the more challenging LiveCodeBench v6. LiveCodeBench (Jain et al., 2025) is continuously updated and uses recent programming problems to reduce data leakage.

Parallel Method	Avg Computational Load		Avg Performance	
<i>Qwen3-0.6B-Base</i>				
Traditional	4.67		44.77	
ADP	1.68	35.9%	45.73	102.15%
<i>Qwen3-1.7B-Base</i>				
Traditional	4.67		61.82	
ADP	1.56	33.4%	65.85	106.53%
<i>Qwen3-4B-Base</i>				
Traditional	3.00		44.45	
ADP	1.47	49.0%	47.03	105.82%

Table 1: Analysis Table of Performance and Computational Load for ADP and Traditional Frameworks. Detailed bifurcated layer and computational load are in Appendix G.

We evaluate in the 0-shot setting.

Math: We use GSM8K (Cobbe et al., 2021) and MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024) to evaluate mathematical performance. GSM8K contains 1K simple problems and tests basic reasoning, whereas MATH-500 contains more challenging problems. We further fine-tune each model on 12,000 examples from MetaMathQA (Yu et al., 2024) to improve alignment with mathematical tasks. We use 5-shot prompting and set the temperature to 0. We test the Math-500 dataset only on Qwen3-4B-Base, as GSM8K is too simple.

Choice Questions: We evaluate on SciQ (Welbl et al., 2017) and HellaSwag (Zellers et al., 2019). SciQ is a multiple-choice science question-answering benchmark, and HellaSwag evaluates commonsense reasoning over physical causality and event prediction. We use 5-shot prompting and select the option with the highest probability among the four choices.

5.3 RQ1: Effectiveness

We analyze the relationship between computational load and performance. The y-axis reports task performance, and the x-axis reports the FLOP ratio for generating one token relative to single-path prefix tuning. Following the scaling behavior observed in ParScale (Chen et al., 2025), we use logarithmic curves to illustrate the performance improvements of ADP at equivalent computational load. Detailed results and layer selection are in Appendix G.

Figures 5a, 5b, and 5c show the code results.

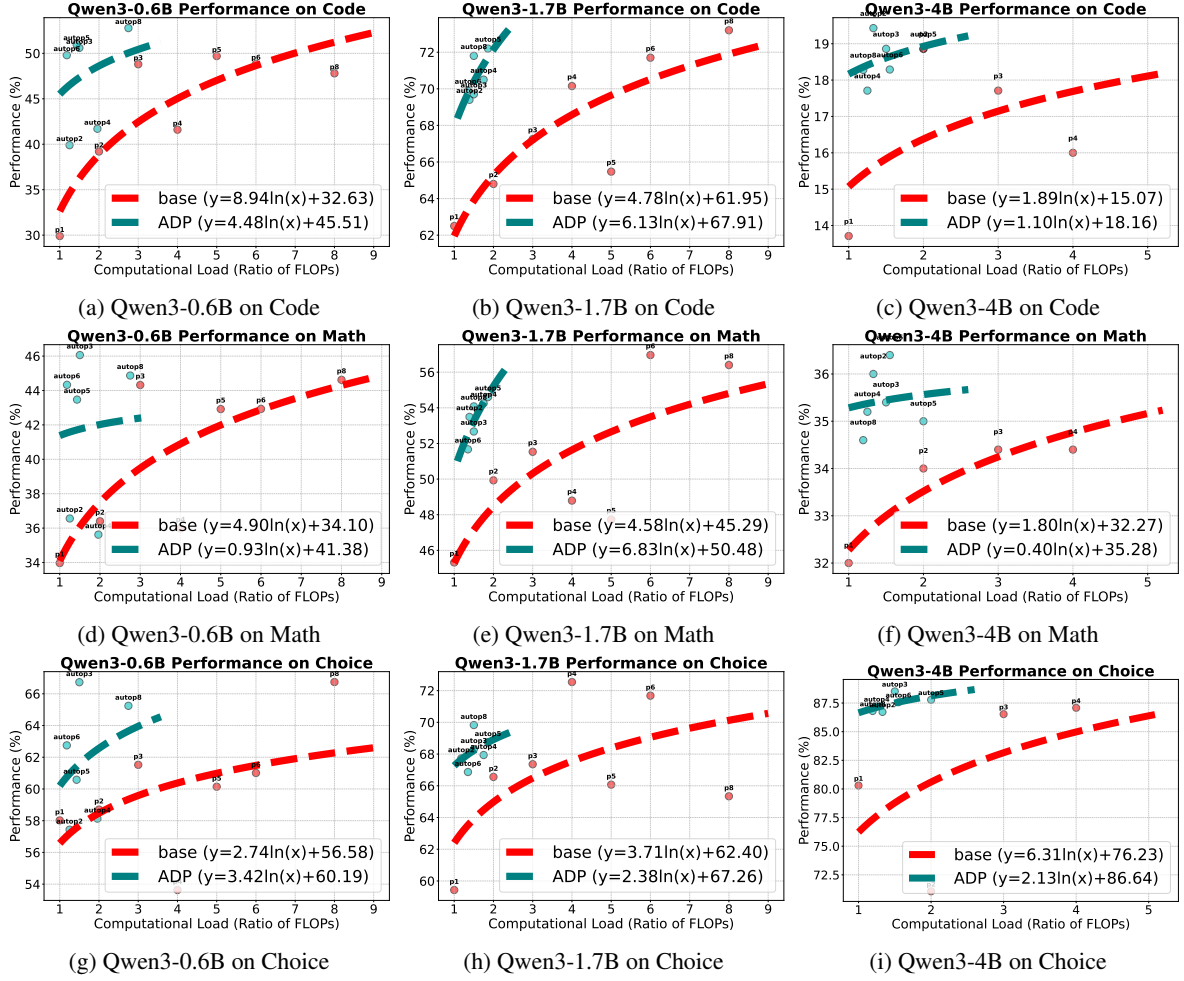


Figure 5: Experimental results. **Teal** denotes ADP, and **Red** denotes the full parallel framework.

ADP improves performance across all three models. At the same computational cost, ADP consistently lies above the full parallel baseline and exhibits a steeper growth rate, demonstrating its effectiveness on code-generation tasks.

Figures 5d, 5e, and 5f show the math results. ADP follows the same trend as on code tasks: it reduces the computational cost of parallel reasoning while preserving the strong reasoning performance of conventional parallel frameworks.

Figures 5g, 5h, and 5i show the results on choice-question benchmarks involving commonsense and logical reasoning. ADP achieves higher performance under the same compute budget and does not degrade relative to ADP variants with the same number of parallel components. These results indicate that, even for tasks with lower reasoning demands, the single-path feature-extraction stage in ADP preserves task-relevant information.

Table 1 further quantifies the computational savings and performance gains of ADP. Overall, ADP

uses only 39.4% of the computation required by conventional parallel frameworks while achieving 104.83% of their performance. Its advantage becomes more pronounced as the degree of parallelism increases. For 8-way parallel models with 0.6B and 1.7B parameters, ADP requires only 26.6% of the computation while reaching 100.39% of the baseline performance.

These results show that ADP substantially reduces computational load without degrading performance. Instead, it improves over conventional parallel frameworks. This improvement stems from its design: a shared prefix focuses on feature extraction, while other prefixes specialize in reasoning and generation. In contrast, conventional parallel frameworks require each branch prefix to perform both feature extraction and inference, which can make these functions difficult to balance even after extensive training. This suggests that ADP enables more efficient training and may be particularly useful in data-scarce settings.

Method	Math	Code	Choice	Avg.	FLOPs
ParScale	21.59	29.80	45.91	32.43	$2.00\times$
ADP	21.20	29.58	47.52	32.77	$1.17\times$

Table 2: Performance and computational load on Gemma-4-E2B with $P = 2$. ADP selects Layer 33 as the bifurcation layer.

Method	Math	Code	Choice	FLOPs
Beam Search (2)	34.52	19.95	58.02	$2.00\times$
Beam Search (4)	35.80	21.15	58.02	$4.00\times$
ParaThinker	38.62	24.95	58.02	$7.24\times$
ParScale	41.20	30.03	60.29	$4.67\times$
ADP	41.82	32.51	61.81	$1.68\times$

Table 3: Comparison with inference-stage parallel baselines on Qwen3-0.6B. FLOPs are normalized by the computational cost of a single path.

To further evaluate whether ADP generalizes beyond the Qwen model family, we conduct additional experiments on Gemma-4-E2B (Team et al., 2026) under the $P = 2$ configuration. ADP automatically selects Layer 33 as the bifurcation layer. As shown in Table 2, ADP achieves an average performance of 32.77, comparable to and slightly higher than the 32.43 of full-stack ParScale, while reducing the computational load from $2.00\times$ to only $1.17\times$. This corresponds to a 41.5% reduction in computation, demonstrating that the effectiveness and efficiency of ADP generalize beyond the Qwen model family.

In addition to ParScale, we compare ADP with Beam Search (Sutskever et al., 2014) and ParaThinker (Wen et al., 2025), which introduce parallel computation at inference time. Although these methods operate under different parallelization paradigms, they provide useful auxiliary baselines for evaluating the performance–compute trade-off. Table 3 reports the results on Qwen3-0.6B. ADP consistently achieves the best performance across mathematics, code, and choice tasks while requiring only $1.68\times$ the FLOPs of a single path. In particular, ADP outperforms ParaThinker while using substantially less computation, and also improves over full-stack ParScale while reducing its computational load from $4.67\times$ to $1.68\times$.

ADP achieves performance improvements across datasets in most models and domains. This demonstrates ADP’s high generalizability, indicating that it can be applied across diverse fields.

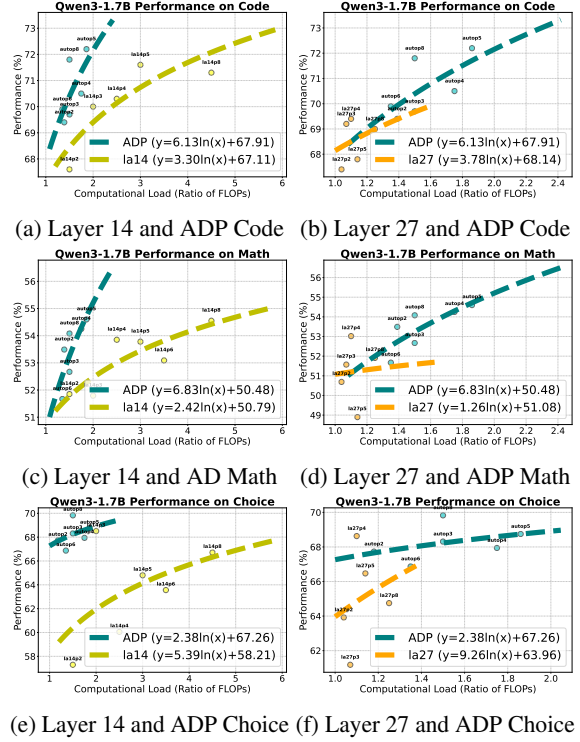


Figure 6: Ablation results. **Teal** denotes ADP, **Yellow** denotes the Layer-14 variant, and **Orange** denotes the Layer-27 variant.

5.4 RQ2: Ablation

To evaluate the adaptive bifurcation-layer selection mechanism, we compare ADP with Section 3. Because these variants differ in computational load, we compare their performance–compute curves. Specifically, we evaluate Qwen3-1.7B-Base with the bifurcation layer fixed at Layer 14 or Layer 27.

Figure 6 shows that ADP consistently outperforms variants with fixed bifurcation layers at Layer 14 or Layer 27. This confirms that similarity-based adaptive selection reliably identifies the transition layer between feature extraction and inference.

These results also support the choice of l^* . Selecting a layer before l^* wastes computation during the expansion stage, whereas selecting a layer after l^* skips part of the contraction stage and degrades performance. Thus, l^* provides a favorable trade-off between efficiency and performance.

5.5 RQ3: Layer Analysis

We examine whether the layers selected by ADP align with inference-start layers identified by prior analyses. Specifically, we analyze Qwen3-1.7B-Base and Qwen3-4B-Base. Following Tan et al. (Tan et al., 2026), we use internal entropy to estimate the inference-start layer. Tan et al. locate

Base-Model	2	3	4	5	6	8
Qwen3-1.7B	17	21	21	22	26	26
Qwen3-4B	24	27	33	27	32	35

Table 4: Layers Automatically Selected by ADP

Training Dataset	Code Alpaca	WizardCoder	MagiCoder
Selected Layer	19	21	20

Table 5: Bifurcation layers selected using different training datasets on Qwen3-0.6B-Base with $P = 2$.

this layer at approximately Layer 24 for Qwen3-4B. Applying the same analysis to Qwen3-1.7B-Base, we identify Layer 20 as the inference-start layer.

Table 4 reports the layers selected by ADP. With few exceptions, ADP selects layers near the boundary between the feature-extraction and inference phases. The exceptions may reflect changes in information processing induced by parallel generation. Overall, these findings support our central hypothesis: similarity follows a unimodal pattern across layers, and cosine similarity can distinguish inference layers from feature-extraction layers. We also find that when the number of parallel paths P increases, the model’s parallel representation dynamics and the joint optimization landscape change significantly. Consequently, a larger P naturally requires more shared computation in early layers to align the paths before they diverge into specific orientations, pushing the optimal l^* deeper.

We further examine the sensitivity of bifurcation-layer selection to the training data. With Qwen3-0.6B-Base and a fixed parallel configuration of $P = 2$, the selected layers on Code Alpaca, WizardCoder (Luo et al., 2024), and Magicoder are 19, 21, and 20, respectively (Table 5). Their narrow range indicates that the selection is relatively robust to the choice of training corpus.

5.6 RQ4: Memory Analysis

We also evaluate the GPU memory usage of Qwen3-1.7B-Base, setting the sequence length to 1024. The experimental results, presented in Table 6, indicate that the average GPU memory usage of ADP is merely 47.5% of that of traditional frameworks—dropping to just 31.7% under 8-way parallelism. This demonstrates that ADP significantly reduces memory; this enables both training and

	1	2	3	4
Full	11.09	17.92	24.15	30.51
ADP	N/A	14.04 78.3%	15.34 63.5%	17.44 57.2%
	5	6	8	Avg
Full	37.56	44.24	57.11	35.25
ADP	19.63 52.3%	15.97 36.1%	18.12 31.7%	16.76 47.5%

Table 6: Memory Usage (GB), Avg represents the average value from 2 to 8.

Method	Speed (tokens/s)	Change
Single Path	138.16	–
ParScale ($P = 2$)	114.50	–17.1%
ADP ($P = 2$)	135.71	–1.7%
ParScale ($P = 4$)	107.37	–22.3%
ADP ($P = 4$)	122.70	–11.1%

Table 7: Wall-clock generation efficiency.

inference on lower-performance devices, thereby enhancing the applicability of ADP.

5.7 RQ5: Wall-clock Efficiency

To evaluate wall-clock efficiency, we measure token generation speed with a batch size of 4, averaged over 20 runs. As shown in Table 7, ADP substantially recovers the throughput loss of Parscale. Under $P = 2$, ADP achieves 135.71 tokens/s versus 114.50 for ParScale and 138.16 for the single-path baseline. Under $P = 4$, ADP improves throughput from 107.37 to 122.70 tokens/s. These results confirm that ADP’s computational savings translate into practical speed improvements.

6 Conclusion

In conclusion, we propose ADP to reduce computational load in parallel frameworks. Leveraging layer-wise functional differences and mathematical analysis, we simplify the parallel feature extraction stage into a single-path design, reducing computational load. Experimental results prove the superiority of ADP. Compared with full-stack parallel frameworks such as ParScale, ADP significantly reduces computational load and memory usage while improving performance.

Limitations

Our current method only performs single-path feature extraction in the free exploration phase, while the inference layers still adopt a fully parallel strategy. This allows for in-depth analysis of changes within the inference layers and further reduction of computational load.

Ethics Statement

Our work complies with the ACL Ethics Policy. All datasets and models are openly accessible. We strictly adhere to the respective licensing conditions of all openly accessible datasets and models, ensuring that any derivative artifacts are used exclusively for research purposes and not deployed commercially. Additionally, all datasets used in our work have been preprocessed by their original curators to remove any identifying information and offensive content. We have not identified any significant ethical considerations associated with our work. We use AI to assist in coding, which has improved efficiency. We also employ AI to create the cartoon avatar of the LLM shown in Figure 1, while all other content was drawn manually. We also release the source code at <https://github.com/ZJU-CTAG/ADP>

Acknowledgements

This research is supported by the National Natural Science Foundation of China (No.92582107), Zhejiang Provincial Natural Science Foundation of China (No.LZ25F020003), and the Fundamental Research Funds for the Central Universities (No. 226-2025-00067).

References

- Josh Alman and Zhao Song. 2024. The fine-grained complexity of gradient computation for training large language models. *Advances in Neural Information Processing Systems*, 37:61419–61454.
- Mouxiong Chen, Binyuan Hui, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Jianling Sun, Junyang Lin, and Zhongxin Liu. 2025. [Parallel scaling law for language models](#). In *Advances in Neural Information Processing Systems*, volume 38, pages 118958–118998. Curran Associates, Inc.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. [Deepseek-v3.2: Pushing the frontier of open large language models](#). *Preprint*, arXiv:2512.02556.
- Lishui Fan, Mouxiong Chen, and Zhongxin Liu. 2025a. [SEK: Self-explained keywords empower large language models for code generation](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6249–6278, Vienna, Austria. Association for Computational Linguistics.
- Lishui Fan, Zhongxin Liu, Haoye Wang, Lingfeng Bao, Xin Xia, and Shanping Li. 2025b. [Fgit: Fault-guided fine-tuning for code generation](#). In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, page 1338–1350. IEEE Press.
- Lishui Fan, Yu Zhang, Mouxiong Chen, and Zhongxin Liu. 2026. Recode: Reinforcing code generation with reasoning-process rewards. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 43896–43914.
- Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, and Yequan Wang. 2025c. [Not all layers of llms are necessary during inference](#). In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pages 5083–5091. International Joint Conferences on Artificial Intelligence Organization. Main Track.
- Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24842–24855.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.
- Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, and 3 others. 2022. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS ’22*, Red Hook, NY, USA. Curran Associates Inc.
- Yi Hu, Xiaojuan Tang, Haotong Yang, and Muhan Zhang. 2024. Case-based or rule-based: how do transformers do the math? In *Proceedings of the*

- 41st International Conference on Machine Learning, pages 19438–19474.
- Li Huang, Zhongxin Liu, Yifan Wu, Tao Yin, Dong Li, Jichao Bi, Nankun Mu, Hongyu Zhang, and Meng Yan. 2026. Deepguard: Secure code generation via multi-layer semantic aggregation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19793–19813.
- Minyoung Huh, Brian Cheung, Tongzhou Wang, and Phillip Isola. 2024. Position: the platonic representation hypothesis. In *Proceedings of the 41st International Conference on Machine Learning, ICML’24*. JMLR.org.
- Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pages 58791–58831.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. [Swe-bench: Can language models resolve real-world github issues?](#) In *International Conference on Learning Representations*, volume 2024, pages 54107–54157.
- Feng Ju, Zeyu Qin, Rui Min, Zhitao He, Lingpeng Kong, and Yi R. Fung. 2026. [Reasoning path divergence: A new metric and curation strategy to unlock llm diverse thinking](#). *Preprint*, arXiv:2510.26122.
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.
- Chengshu Li, Jacky Liang, Fei Xia, Andy Zeng, Sergey Levine, Dorsa Sadigh, Karol Hausman, Xinyun Chen, Li Fei-Fei, et al. 2023. Chain of code: Reasoning with a language model-augmented code interpreter. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.
- Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597.
- Baohao Liao, Yuhui Xu, Hanze Dong, Junnan Li, Christof Monz, Silvio Savarese, Doyen Sahoo, and Caiming Xiong. 2025. Reward-guided speculative decoding for efficient llm reasoning. In *International Conference on Machine Learning*, pages 37555–37572. PMLR.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. [Let’s verify step by step](#). In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.
- Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. [Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 21558–21572. Curran Associates, Inc.
- Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.
- Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2024. [Wizardcoder: Empowering code large language models with evol-instruct](#). In *International Conference on Learning Representations*, volume 2024, pages 27168–27188.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.
- Xinyuan Song, Keyu Wang, PengXiang Li, Lu Yin, and Shiwei Liu. 2026. Demystifying the roles of llm layers in retrieval, knowledge, and reasoning. In *ICASSP 2026-2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 16792–16796. IEEE.
- Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.
- Yuqiao Tan, Minzheng Wang, Shizhu He, Huanxuan Liao, Chengfeng Zhao, Qiunan Lu, Tian Liang, Jun Zhao, and Kang Liu. 2026. [Bottom-up policy optimization: Your language model policy secretly contains internal policies](#). *Preprint*, arXiv:2512.19673.
- Yao Tang, Li Dong, Yaru Hao, Qingxiu Dong, Furu Wei, and Jiatao Gu. 2026. [Multiplex thinking: Reasoning via token-wise branch-and-merge](#). *Preprint*, arXiv:2601.08808.
- Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, Mayank Chaturvedi, Aditya Chawla, Victor Cotruta, Alice Coucke, Phil Culliton, Robert Dadashi, Lucas Dixon, Mohamed Elhawaty, Utku Evci, and 304 others. 2026. [Gemma 4 technical report](#). *Preprint*, arXiv:2607.02770.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2024. Magicoder: empowering code generation with oss-instruct. In *Proceedings of the 41st International Conference on Machine Learning*, ICML’24. JMLR.org.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. 2017. [Crowdsourcing multiple choice science questions](#). In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 94–106, Copenhagen, Denmark. Association for Computational Linguistics.

Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. 2025. [Parathinker: Native parallel thinking as a new paradigm to scale llm test-time compute](#). *Preprint*, arXiv:2509.04475.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. [Qwen3 technical report](#). *Preprint*, arXiv:2505.09388.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. Metamath: Bootstrap your own mathematical questions for large language models. In *International Conference on Learning Representations*, volume 2024, pages 45040–45061.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Kunhao Zheng, Juliette Decugis, Jonas Gehring, Taco Cohen, Benjamin Negrevergne, and Gabriel Synnaeve. 2025. [What makes large language models reason in \(multi-turn\) code generation?](#) In *International Conference on Learning Representations*, volume 2025, pages 40144–40181.

A Experimental Proof of Assumption A

We use the relative ℓ_2 distance between hidden states, denoted as `hidden_rel_l2`, to quantify the representation-level discrepancy between the target model and the layer-0-start baseline. Given the hidden states H_l^{target} and H_l^{base} at layer l , the metric is defined as

$$\text{hidden_rel_l2}(l) = \frac{\|H_l^{\text{target}} - H_l^{\text{base}}\|_2}{\|H_l^{\text{base}}\|_2 + \epsilon},$$

where ϵ is a small constant for numerical stability. This metric measures the magnitude of the hidden-state deviation after normalizing by the scale of the baseline representation. A smaller value indicates that the target model produces hidden representations closer to those of the baseline model, while a value of zero corresponds to identical hidden states. Therefore, a decreasing `hidden_rel_l2` across layers suggests that the target model progressively aligns with the layer-0-start baseline in representation space.

Method	Start	End	Change
0.6B P2	0.753235	0.722918	-0.0303171
0.6B P4	0.855034	0.723417	-0.131616
0.6B P8	1.54065	0.748282	-0.792369
1.7B P2	0.87684	0.598589	-0.27825
1.7B P4	0.774145	0.472126	-0.302019
1.7B P8	0.772808	0.677858	-0.0949494
4B P2	1.04937	0.814684	-0.234683
4B P4	0.850918	0.682198	-0.168721

Table 8: Changes between ADP and the traditional framework lie in the distance between the hidden states at the start and end of ADP’s parallelism. **B** means the size of the model and **P** means the number of parallel.

Table 8 empirically supports the validity of Assumption 1 that after ADP starts to run in parallel, ρ is less than 1, and ADP will continuously move closer to the traditional framework.

We further illustrate the layer-by-layer changes of `hidden_rel_l2` in ADP. The results are shown in Tables 9 and 10, which further demonstrate the validity of Assumption 1.

Layer	hidden_rel_l2
21	0.87684
22	0.726789
23	0.690886
24	0.654288
25	0.653256
26	0.644326
27	0.644578
28	0.598589

Table 9: Qwen3-1.7B Layer-by-layer Hidden_rel_l2 Changes Results under P2 Conditions.

Layer	hidden_rel_l2
33	0.850918
34	0.787046
35	0.792271
36	0.682198

Table 10: Qwen3-4B Layer-by-layer Hidden_rel_l2 Changes Results under P4 Conditions.

B Proof of Similarity Changes

We consider raw hidden states $h_l^{(i)} \in \mathbb{R}^d$ for $i = 1, \dots, p$, and define the corresponding unit directions and their empirical mean as

$$q_l^{(i)} = \frac{h_l^{(i)}}{\|h_l^{(i)}\|},$$

$$\bar{q}_l = \frac{1}{p} \sum_{i=1}^p q_l^{(i)}.$$

The average pairwise cosine similarity at layer l is

$$\bar{C}_l = \frac{2}{p(p-1)} \sum_{i < j} \langle q_l^{(i)}, q_l^{(j)} \rangle.$$

We also define the directional variance

$$D_l = \frac{1}{p} \sum_{i=1}^p \|q_l^{(i)} - \bar{q}_l\|^2.$$

Let the residual update from layer l to layer $l+1$ be $u_l^{(i)} = h_{l+1}^{(i)} - h_l^{(i)}$. The induced directional

increment and its empirical mean are

$$w_l^{(i)} = q_{l+1}^{(i)} - q_l^{(i)}$$

$$= \frac{h_l^{(i)} + u_l^{(i)}}{\|h_l^{(i)} + u_l^{(i)}\|} - \frac{h_l^{(i)}}{\|h_l^{(i)}\|},$$

$$\bar{w}_l = \frac{1}{p} \sum_{i=1}^p w_l^{(i)}.$$

We decompose the centered directions and centered directional updates as

$$\delta_l^{(i)} = q_l^{(i)} - \bar{q}_l, \quad \eta_l^{(i)} = w_l^{(i)} - \bar{w}_l.$$

Here, $\delta_l^{(i)}$ is the pre-existing directional deviation at layer l , whereas $\eta_l^{(i)}$ is the prefix-dependent component of the directional update from layer l to layer $l+1$.

Since $\|q_l^{(i)}\| = 1$, the variance satisfies

$$D_l = \frac{1}{p} \sum_{i=1}^p \|q_l^{(i)} - \bar{q}_l\|^2$$

$$= 1 - \|\bar{q}_l\|^2.$$

Moreover,

$$\left\| \sum_{i=1}^p q_l^{(i)} \right\|^2 = p + 2 \sum_{i < j} \langle q_l^{(i)}, q_l^{(j)} \rangle,$$

and therefore

$$p^2 \|\bar{q}_l\|^2 = p + 2 \sum_{i < j} \langle q_l^{(i)}, q_l^{(j)} \rangle.$$

Substituting this identity into the definition of $\bar{C}_l$ gives

$$\bar{C}_l = \frac{p \|\bar{q}_l\|^2 - 1}{p-1}$$

$$= 1 - \frac{p}{p-1} D_l.$$

Next, using $q_{l+1}^{(i)} = q_l^{(i)} + w_l^{(i)}$, we have

$$\bar{q}_{l+1} = \bar{q}_l + \bar{w}_l.$$

Thus,

$$q_{l+1}^{(i)} - \bar{q}_{l+1} = \delta_l^{(i)} + \eta_l^{(i)}.$$

Define

$$\Phi_l = 2 \sum_{i=1}^p \langle \delta_l^{(i)}, \eta_l^{(i)} \rangle + \sum_{i=1}^p \|\eta_l^{(i)}\|^2.$$

Then

$$\begin{aligned} D_{l+1} &= \frac{1}{p} \sum_{i=1}^p \left\| \delta_l^{(i)} + \eta_l^{(i)} \right\|^2 \\ &= D_l + \frac{1}{p} \Phi_l. \end{aligned}$$

Combining this expression with $\bar{C}_l = 1 - \frac{p}{p-1} D_l$ yields

$$\begin{aligned} \bar{C}_{l+1} - \bar{C}_l &= -\frac{p}{p-1} (D_{l+1} - D_l) \\ &= -\frac{1}{p-1} \Phi_l. \end{aligned}$$

Consequently,

$$\begin{aligned} \bar{C}_{l+1} < \bar{C}_l &\iff \Phi_l > 0, \\ \bar{C}_{l+1} > \bar{C}_l &\iff \Phi_l < 0. \end{aligned}$$

A decrease in similarity therefore occurs when the prefix-dependent directional updates $\eta_l^{(i)}$ amplify the existing deviations $\delta_l^{(i)}$. Conversely, similarity increases when these updates are sufficiently aligned against the existing deviations, so that the negative cross term dominates the positive quadratic term $\sum_{i=1}^p \|\eta_l^{(i)}\|^2$.

C Proof of the Best Bifurcated Layer

For a switch layer s , define the directional error between the ADP trajectory and the full-parallel trajectory at layer l as

$$E_l^q(s) = \max_{1 \leq i \leq p} \left\| q_l^{\text{ADP},(i)}(s) - q_l^{\text{full},(i)} \right\|.$$

To relate the error at the switch layer to the final-layer error, we use the following local stability condition.

Directional error contraction. For every $l \geq l^*$, there exists a constant $0 \leq \rho_l < 1$ such that, within a local region containing both the ADP trajectory and the full-parallel trajectory,

$$E_{l+1}^q(s) \leq \rho_l E_l^q(s).$$

Applying this contraction recursively from l^* to the final layer L gives

$$E_L^q(l^*) \leq E_{l^*}^q(l^*) \prod_{k=l^*}^{L-1} \rho_k.$$

More generally, for any switch layer $s \geq l^*$, the same argument yields

$$E_L^q(s) \leq E_s^q(s) \prod_{k=s}^{L-1} \rho_k.$$

Hence a later switch layer s has no larger final directional-error bound than l^* whenever

$$\begin{aligned} E_s^q(s) \prod_{k=s}^{L-1} \rho_k &\leq E_{l^*}^q(l^*) \prod_{k=l^*}^{L-1} \rho_k \\ \iff E_s^q(s) &\leq E_{l^*}^q(l^*) \prod_{k=l^*}^{s-1} \rho_k. \end{aligned}$$

This establishes the claimed comparison criterion for the bifurcation layer.

It remains to translate the final directional error into a loss difference. For cross-entropy loss,

$$\nabla_z \ell(z, y) = \text{softmax}(z) - e_y.$$

Because both $\text{softmax}(z)$ and e_y lie in the probability simplex,

$$\|\nabla_z \ell(z, y)\|_2 \leq \sqrt{2}.$$

By the mean value theorem,

$$|\ell(z_1, y) - \ell(z_2, y)| \leq \sqrt{2} \|z_1 - z_2\|_2.$$

For logits of the form $z = Wq + b$, we also have

$$\|z_1 - z_2\|_2 \leq \|W\|_2 \|q_1 - q_2\|_2.$$

Therefore, for each path i ,

$$\begin{aligned} &\left| \ell(z_{\text{ADP}}^{(i)}, y) - \ell(z_{\text{full}}^{(i)}, y) \right| \\ &\leq \sqrt{2} \|W\|_2 \left\| q_L^{\text{ADP},(i)}(s) - q_L^{\text{full},(i)} \right\|_2 \\ &\leq \sqrt{2} \|W\|_2 E_L^q(s). \end{aligned}$$

Averaging over $i = 1, \dots, p$ gives

$$|\mathcal{L}_{\text{ADP}} - \mathcal{L}_{\text{full}}| \leq \sqrt{2} \|W\|_2 E_L^q(s).$$

D Detailed Training Settings

The prefix length is set to 24, and the prefix-generating MLP has two layers. The batch size is set to 256. We set T to 200 steps for the warm-up pretraining phase. We use Adam as the optimizer, and set the context length to 1024. The β parameter of Adam is $\beta = (0.9, 0.95)$ and the remaining parameters are PyTorch’s defaults. The total training step is 3950.

Algorithm 1: ADP: Discovery and Transition Strategy

```

1 for  $step\ t = 1$  to  $T_{total}$  do
2   if  $t \leq T$  then
3     Train  $M$  in conventional mode
4     if  $t = T$  then
5        $l^* \leftarrow \arg \min_{l \in [2, L-1]} \bar{S}^{(l)}$ 
6       // Locate minimum similarity
7        $M \leftarrow$ 
8       TransformToBifurcated( $M, l^*$ )
9   else
10    Reinitialize the prefix and learning rate
11    Train  $M$  in adaptive mode
12    // Single trunk,  $P$  heads
13 return  $M$  for ADP
  
```

During training, we warm up for 100 steps at a learning rate of 10^{-4} and then apply cosine annealing to 10^{-6} .

Our training and inference GPUs are NVIDIA H100 80GB HBM3. Our CPU is Intel(R) Xeon(R) Platinum 8558.

We test Pass@3 with the sampling temperature set to 0.7°C. Pass@3 is a metric that measures whether a correct solution is generated in three attempts. The formula is

$$\text{Pass@3} = \frac{1}{N} \sum_{i=1}^N \mathbf{1} \left\{ \bigvee_{j=1}^3 s_{i,j} \text{ passes} \right\}$$

The ‘max new tokens’ value was set to 512 for the test code dataset and to 256 for the test mathematical dataset.

E Pseudocode of ADP

Algorithm 1 presents the pseudocode for the ADP method. We first perform a warm-up pretraining phase of T steps. Subsequently, we re-initialize the prefix and learning rate, and switch to the ADP framework to conduct efficient training. During the inference phase, we directly utilize the ADP framework to perform efficient inference.

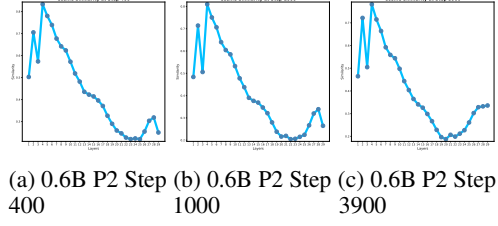


Figure 7: Evolution of inter-path cosine similarity during training on Qwen3-0.6B-Base Parallel 2.

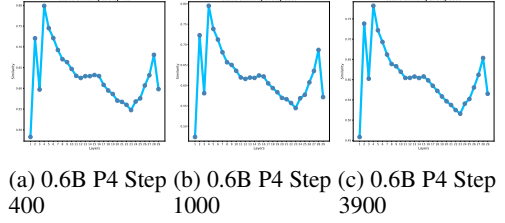


Figure 8: Evolution of inter-path cosine similarity during training on Qwen3-0.6B-Base Parallel 4.

F Stability of Bifurcation-Layer Selection

F.1 More cosine similarity variation curves

Figures 7, 8, 9, 10, 11, and 12 demonstrate that the similarity is unimodal and the position of the minimum value remains unchanged.

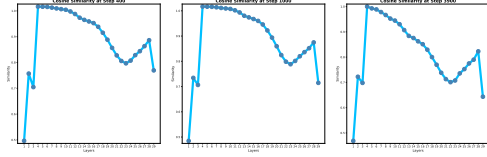
F.2 Stability Under Different Seeds

We further examine the stability of the bifurcation-layer selection under a fixed parallel configuration P . Using Qwen3-1.7B-Base with $P = 2$, we track the selected layer throughout training. As shown in Table 11, the selected layer quickly converges to Layer 17 within the first 150 steps and remains unchanged for the rest of training.

We also evaluate the robustness of the selection across different random seeds. Using three seeds (123, 456, and 99) under the same $P = 2$ configuration, ADP consistently selects Layer 17 as the bifurcation layer. These results show that, with P fixed, the selected bifurcation layer is stable across both training progress and random initialization, supporting the use of a short warm-up stage to determine l^* .

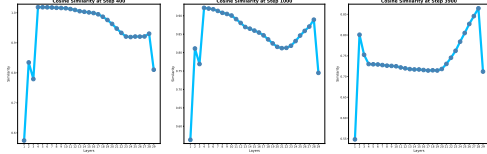
G Detailed Experimental Results

Table 12, 13, 14, 15, 16, and 17 presents the performance of the detailed experimental results.



(a) 1.7B P2 Step 400 (b) 1.7B P2 Step 1000 (c) 1.7B P2 Step 3900

Figure 9: Evolution of inter-path cosine similarity during training on Qwen3-1.7B-Base Parallel 2.



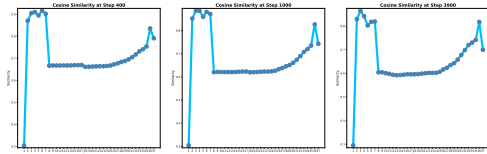
(a) 1.7B P4 Step 400 (b) 1.7B P4 Step 1000 (c) 1.7B P4 Step 3900

Figure 10: Evolution of inter-path cosine similarity during training on Qwen3-1.7B-Base Parallel 4.



(a) 4B P2 Step 400 (b) 4B P2 Step 1000 (c) 4B P2 Step 3900

Figure 11: Evolution of inter-path cosine similarity during training on Qwen3-4B-Base Parallel 2.



(a) 4B P4 Step 400 (b) 4B P4 Step 1000 (c) 4B P4 Step 3900

Figure 12: Evolution of inter-path cosine similarity during training on Qwen3-4B-Base Parallel 4.

Step	50	100	150	200	300
Selected Layer	21	19	17	17	17
Step	500	1000	2000	3000	3900
Selected Layer	17	17	17	17	17

Table 11: Stability of the selected bifurcation layer across training steps on Qwen3-1.7B-Base with $P = 2$.

p	Method	l^*	Choice				Math				FLOP
			SciQ	HellaSwag	Avg.	Δ	GSM	MATH	Avg.	Δ	
1	Full	–	82.3	33.73	58.02	–	39.50	28.44	33.97	–	1
2	Full	–	82.9	34.50	58.70	–	43.67	29.13	36.40	–	2
	ADP	21	81.8	33.06	57.43	–1.27	44.12	28.99	36.56	+0.16	1.25
3	Full	–	84.7	38.34	61.52	–	57.24	31.40	44.32	–	3
	ADP	21	92.9	40.56	66.73	+5.21	59.67	32.45	46.06	+1.74	1.50
4	Full	–	75.7	31.55	53.63	–	43.67	28.31	35.99	–	4
	ADP	21	83.1	33.15	58.13	+4.50	42.53	28.71	35.62	–0.37	1.96
5	Full	–	84.4	35.87	60.14	–	58.23	27.60	42.92	–	5
	ADP	25	85.2	35.95	60.58	+0.44	58.53	28.40	43.74	+0.55	1.43
6	Full	–	84.8	37.22	61.01	–	56.86	29.00	42.93	–	6
	ADP	27	86.0	39.50	62.75	+1.74	60.65	28.00	44.33	+1.40	1.18
8	Full	–	92.9	40.57	66.74	–	58.98	30.26	44.62	–	8
	ADP	21	92.6	37.88	65.24	–1.50	58.30	31.43	44.87	+0.25	2.75

Table 12: Choice and mathematical reasoning results of Qwen3-0.6B under different parallel decoding sizes. p denotes the number of parallel candidates. l^* denotes the bifurcation layer selected by ADP. Full is the full parallel baseline, while ADP is our adaptive parallel decoding method. Δ denotes the average-score difference between ADP and Full under the same p . FLOP represents the ratio of FLOPs to those of a single-path frame.

p	Method	l^*	Code, $T = 0$				Code, $T = 0.7$				FLOP
			HE	MBPP	Avg.	Δ	HE	MBPP	Avg.	Δ	
1	Full	–	23.8/20.7	10.1/8.7	15.83	–	32.9/28.0	32.5/26.2	29.90	–	1
2	Full	–	29.3/26.8	31.0/26.5	28.15	–	33.5/28.7	51.3/43.1	39.20	–	2
	ADP	21	26.8/23.8	20.4/17.2	22.05	–6.10	31.7/28.0	46.8/39.7	36.60	–2.60	1.25
3	Full	–	28.0/26.2	41.0/34.9	32.53	–	47.6/41.5	56.9/49.2	48.80	–	3
	ADP	21	36.0/30.5	44.2/36.2	36.73	+4.20	47.0/40.2	62.2/52.9	50.60	+1.80	1.50
4	Full	–	28.0/24.4	28.3/23.0	25.93	–	37.2/33.5	52.9/42.6	41.60	–	4
	ADP	21	26.2/23.2	26.7/23.3	24.85	–1.08	37.2/31.7	54.8/46.8	41.70	+0.10	1.96
5	Full	–	30.5/26.2	28.6/24.9	27.55	–	48.2/42.7	57.9/50.0	49.70	–	5
	ADP	25	31.7/29.9	46.0/40.5	37.03	+9.48	47.0/39.6	63.0/54.5	51.00	+1.30	1.43
6	Full	–	28.7/25.6	39.7/35.7	32.43	–	40.9/35.4	63.5/55.6	48.90	–	6
	ADP	27	32.3/29.3	51.3/44.7	39.40	+6.97	42.1/37.2	64.3/55.6	49.80	+0.90	1.18
8	Full	–	26.8/25.0	46.3/36.2	33.58	–	39.6/33.5	63.5/54.8	47.80	–	8
	ADP	21	32.3/29.3	41.8/36.5	34.98	+1.40	47.0/41.5	64.3/52.4	51.30	+3.50	2.75

Table 13: Code generation results of Qwen3-0.6B under different parallel decoding sizes. l^* denotes the bifurcation layer selected by ADP. For HumanEval (HE) and MBPP, each cell reports Base/Plus scores from EvalPlus. Δ denotes the average-score difference between ADP and Full under the same p . FLOP represents the ratio of FLOPs to those of a single-path frame.

p	Method	l^*	Choice				Math				FLOP
			SciQ	HellaSwag	Avg.	Δ	GSM	MATH	Avg.	Δ	
1	Full	–	85.2	36.35	59.43	–	57.24	33.4	45.32	–	1
2	Full	–	87.3	45.82	66.56	–	65.66	34.2	49.93	–	2
	ADP	17	89.8	45.66	67.73	+1.17	69.98	37.0	53.49	+3.56	1.39
3	Full	–	90.5	44.22	67.36	–	69.45	33.6	51.53	–	3
	ADP	21	90.2	46.41	68.31	+0.95	69.14	36.2	52.67	+1.14	1.50
4	Full	–	91.6	53.48	72.54	–	62.17	35.4	48.79	–	4
	ADP	21	89.6	46.27	67.94	−4.60	71.10	37.4	54.25	+5.46	1.75
5	Full	–	89.9	42.23	66.07	–	63.46	32.0	47.73	–	5
	ADP	22	91.1	46.39	68.75	+2.68	71.19	38.0	54.60	+6.87	1.86
6	Full	–	91.5	51.85	71.68	–	75.51	38.4	56.96	–	6
	ADP	26	89.9	43.84	66.87	−4.81	68.54	34.8	51.67	−5.29	1.35
8	Full	–	90.1	40.57	65.34	–	76.19	36.6	56.40	–	8
	ADP	26	90.8	48.86	69.83	+4.49	71.95	36.2	54.08	−2.32	1.50

Table 14: Choice and mathematical reasoning results of Qwen3-1.7B under different parallel decoding sizes. p denotes the number of parallel candidates. l^* denotes the bifurcation layer selected by ADP. Full is the full parallel baseline, while ADP is our adaptive parallel decoding method. Δ denotes the average-score difference between ADP and Full under the same p . FLOP represents the ratio of FLOPs to those of a single-path frame.

p	Method	l^*	Code, $T = 0$				Code, $T = 0.7$				FLOP
			HE	MBPP	Avg.	Δ	HE	MBPP	Avg.	Δ	
1	Full	–	61.6/54.3	62.4/52.4	57.68	–	59.1/54.9	72.8/63.2	62.50	–	1
2	Full	–	59.1/53.7	66.1/54.8	58.43	–	62.8/56.7	74.6/65.1	64.80	–	2
	ADP	17	59.8/55.5	54.2/46.0	53.88	−4.55	70.7/66.5	75.7/64.8	69.42	+4.62	1.39
3	Full	–	59.8/53.0	65.9/55.8	58.63	–	68.3/62.8	73.3/64.6	67.25	–	3
	ADP	21	58.5/53.0	68.3/57.7	59.38	+0.75	72.6/66.5	75.7/65.6	70.10	+2.85	1.50
4	Full	–	67.1/60.4	48.1/40.7	54.08	–	73.2/64.0	76.5/66.9	70.15	–	4
	ADP	21	59.1/54.3	67.2/56.6	59.30	+5.22	71.3/65.2	81.0/69.3	71.70	+1.55	1.75
5	Full	–	57.3/51.8	65.6/56.1	57.70	–	64.0/59.0	75.4/63.5	65.47	–	5
	ADP	22	62.2/56.7	65.1/55.6	59.9	+2.20	73.2/68.9	79.1/67.5	72.18	+6.71	1.86
6	Full	–	67.1/59.1	62.2/52.4	60.20	–	73.2/65.9	80.2/67.5	71.70	–	6
	ADP	26	59.1/53.7	61.1/52.9	56.70	−3.50	72.0/67.1	75.9/65.9	70.22	−1.48	1.35
8	Full	–	68.3/60.4	63.0/52.6	61.08	–	74.4/69.5	79.9/69.0	73.20	–	8
	ADP	26	61.0/56.7	66.1/56.3	60.03	−1.05	75.0/68.3	76.7/67.2	71.80	−1.40	1.50

Table 15: Code generation results of Qwen3-1.7B under different parallel decoding sizes. l^* denotes the bifurcation layer selected by ADP. For HumanEval (HE) and MBPP, each cell reports Base/Plus scores from EvalPlus. Δ denotes the average-score difference between ADP and Full under the same p . FLOP represents the ratio of FLOPs to those of a single-path frame.

p	Method	l^*	Choice				Math		FLOP
			SciQ	HellaSwag	Avg.	Δ	MATH	Δ	
1	Full	–	90.5	70.10	80.30	–	32.00	–	1
2	Full	–	65.6	76.52	71.06	–	34.00	–	2
	ADP	24	94.5	78.93	86.72	+15.66	36.00	+2.00	1.33
3	Full	–	94.5	78.56	86.53	–	34.40	–	3
	ADP	27	94.8	82.23	88.52	+1.99	35.40	+1.00	1.50
4	Full	–	94.5	79.66	87.08	–	34.40	–	4
	ADP	33	94.8	79.61	87.21	+0.13	35.20	+0.80	1.25
5	ADP	27	94.8	80.76	87.78	–	35.00	–	2.00
6	ADP	32	94.8	80.34	87.57	–	36.40	–	1.55
8	ADP	35	94.8	78.79	86.80	–	34.60	–	1.19

Table 16: Choice and mathematical reasoning results of Qwen3-4B under different parallel decoding sizes. p denotes the number of parallel candidates. l^* denotes the bifurcation layer selected by ADP. Full is the full parallel baseline, while ADP is our adaptive parallel decoding method. Since GSM8K is removed due to task difficulty, the math results only report MATH. Δ denotes the difference between ADP and Full under the same p . FLOP represents the ratio of FLOPs to those of a single-path frame.

p	Method	l^*	LiveCodeBench v6	Δ	FLOP
1	Full	–	13.71	–	1
2	Full	–	18.86	–	2
	ADP	24	19.43	+0.57	1.33
3	Full	–	17.71	–	3
	ADP	27	18.86	+1.15	1.50
4	Full	–	16.00	–	4
	ADP	33	17.71	+1.71	1.25
5	ADP	27	18.86	–	2.00
6	ADP	32	18.29	–	1.55
8	ADP	35	18.28	–	1.19

Table 17: Code generation results of Qwen3-4B under different parallel decoding sizes. The code benchmark is LiveCodeBench v6. p denotes the number of parallel candidates. Full is the full parallel baseline, while ADP is our adaptive parallel decoding method. Δ denotes the difference between ADP and Full under the same p . FLOP represents the ratio of FLOPs to those of a single-path frame.

p	Method	l^*	Choice			Code			Math			All Avg.
			SciQ	HellaSwag	Avg.	HE	MBPP	Avg.	GSM	MATH	Avg.	
2	Full	–	65.6	26.22	45.91	23.2/20.1	41.0/34.9	29.80	32.98	10.20	21.59	32.43
	ADP	33	67.1	27.93	47.52	27.4/22.6	37.3/31.0	29.58	30.40	12.00	21.20	32.77

Table 18: Results of Gemma-4-E2B under $p = 2$. l^* denotes the bifurcation layer selected by ADP. For HumanEval (HE) and MBPP, each cell reports Base/Plus scores. All Avg. is the average over the three task domains.