

# ASPD: UNLOCKING ADAPTIVE SERIAL-PARALLEL DECODING BY EXPLORING INTRINSIC PARALLELISM IN LLMs

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

The increasing scale and complexity of large language models (LLMs) pose significant inference latency challenges, primarily due to their autoregressive decoding paradigm characterized by the sequential nature of next-token prediction. By re-examining the outputs of autoregressive models, we observed that some segments exhibit parallelizable structures, which we term intrinsic parallelism. Decoding each parallelizable branch simultaneously (*i.e.* parallel decoding) can significantly improve the overall inference speed of LLMs. In this paper, we propose an **Adaptive Serial-Parallel Decoding (ASPD)**, which addresses two core challenges: automated construction of parallelizable data and efficient parallel decoding mechanism. More specifically, we introduce a non-invasive pipeline that automatically extracts and validates parallelizable structures from the responses of autoregressive models. To empower efficient adaptive serial-parallel decoding, we implement a **Hybrid Decoding Engine** which enables seamless transitions between serial and parallel decoding modes while maintaining a reusable KV cache, maximizing computational efficiency. Extensive evaluations across General Tasks, Retrieval-Augmented Generation and Mathematical Reasoning demonstrate that **ASPD** achieves unprecedented performance in both effectiveness and efficiency. Notably, on Vicuna Bench, our method achieves up to 3.10x speedup (1.82x on average) while maintaining response quality within 1% difference compared to autoregressive models, realizing significant acceleration without compromising generation quality. The source code is available for review at an anonymous repository: <https://anonymous.4open.science/r/ASPD>.

## 1 INTRODUCTION

Recent advances in large language models (LLMs) (Yang et al., 2025a; Seed et al., 2025) have dramatically increased both model size and context length. While these improvements enhance model capabilities, they also increase inference latency due to the sequential nature of autoregressive decoding, posing challenges for practical applications that require low latency. Our analysis of LLM outputs reveals a key insight: despite being decoded sequentially, many model responses contain inherent parallelism that can be leveraged for parallel decoding. As illustrated in Figure 1, across various scenarios including general dialogue (LMSYS, 2023), our internal machine reading comprehension (MRC) benchmarks, Retrieval-Augmented Generation (Neural Bridge AI, 2023), and mathematical reasoning (Hugging Face, 2025), model responses consistently reveal significant potential for parallelization. By harnessing these inherently parallelizable segments for concurrent output, decoding speed can be substantially accelerated.

However, this also raises several fundamental technical challenges: First, identifying parallelizable segments while preserving semantic integrity is inherently difficult, due to the intricate dependencies present in natural language generation. Second, it is critical to ensure strict independence across parallel branches: each must remain contextually isolated during decoding, yet collectively yield a coherent output when merged. Third, the sophisticated coordination of positional information across parallel branches presents significant architectural challenges, particularly in preserving proper token relations and temporal coherence throughout parallel generation.

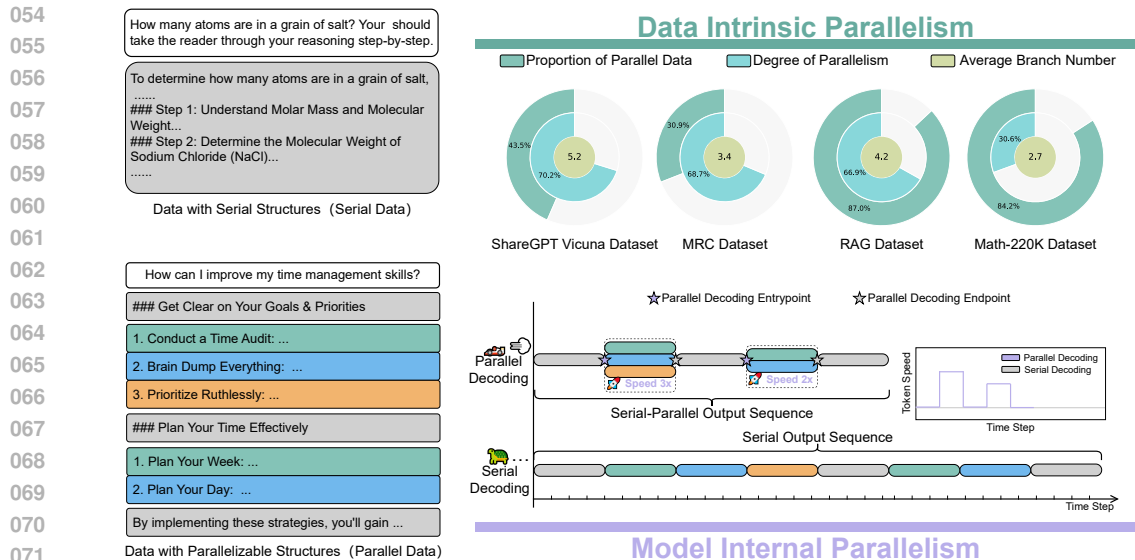


Figure 1: Overview of data intrinsic and model internal parallelism. The definition of Proportion of Parallel Data, Degree of Parallelism and Average Branch Number can be found in Section 4.1.

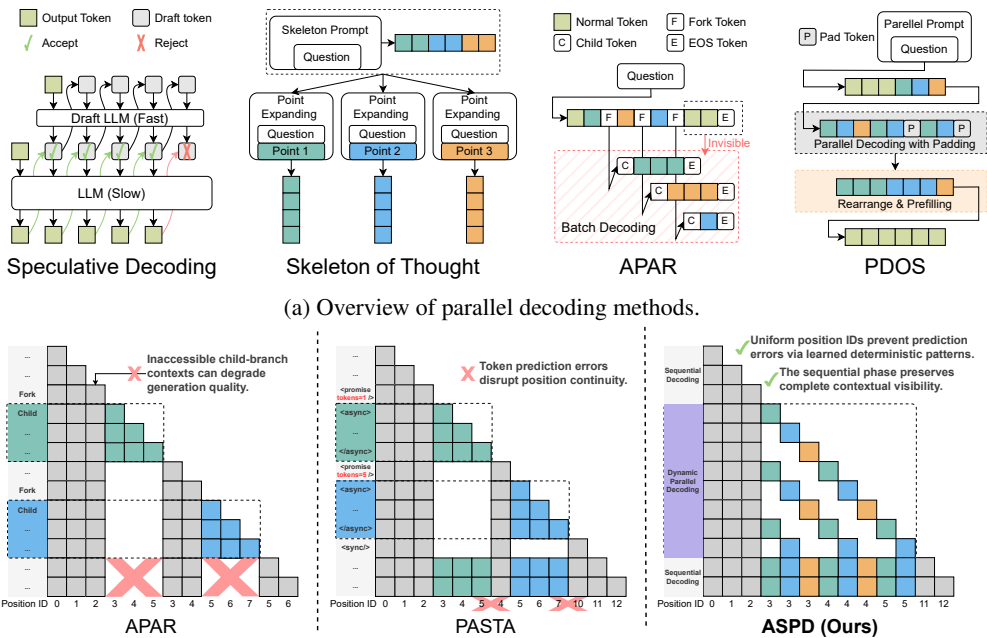
Recent attempts to address these challenges have yielded promising but limited solutions. APAR (Liu et al., 2024) discarded the KV-caches of parallel branches during generation to address position encoding issues, while compromising response quality. PASTA (Jin et al., 2025) explored asynchronous parallel decoding with pre-allocated position ranges, but struggles with position encoding mismatches of actual generation lengths.

Addressing these challenges is essential for unlocking the full potential of parallel decoding in practical LLM systems. To better exploit the intrinsic parallelism within autoregressive models, we present Adaptive Serial-Parallel Decoding, a novel framework that more effectively harnesses the model’s inherent parallel capabilities through a dual-perspective optimization of data utilization and architectural innovation. Our approach first extracts inherent parallelism patterns from model responses to serve as training corpora for parallelization. We further propose an internal parallelization module that enables parallel processing in one go. By designing branch-specific attention masks and consistent positional ids across parallel branches, our method ensures that: (1) parallel branch generation maintains behavioral consistency with native serial decoding from each branch’s perspective; and (2) upon completion of all parallel branches, switching back to the primary branch incurs no information loss and recomputation overhead. To achieve these objectives, we propose a Hybrid Decoding Engine that supports efficient parallel decoding and iterative serial-parallel decoding.

Our key contributions include:

- We develop an innovative non-invasive pipeline that automatically discovers and extracts inherent parallelizable structures from autoregressive model responses, which identifies semantically independent components that can be processed concurrently while preserving the response’s original style. This enables us to build high-quality parallel training corpora automatically without altering the response probability distribution.
- We introduces a novel internal parallelization architecture that combines custom branch-invisible masking and branch-shared position embedding, enabling efficient parallel processing and seamless integration of parallel branches without batching or threading overhead. Based on this architecture, our proposed Hybrid Decoding Engine achieves efficient iterative serial-parallel decoding.
- Through comprehensive evaluation across diverse benchmarks - including general tasks (Vicuna Bench, MT Bench), retrieval-augmented generation (Neural-Bridge-RAG), and mathematical reasoning (MATH500, AMC23, GPQA, AIME2024, AIME2025) - we demonstrate significant improvements in both effectiveness and efficiency compared to existing approaches, achieving an optimal balance between these two critical metrics.

108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161



(a) Overview of parallel decoding methods.

(b) Internal parallel decoding mechanisms.

Figure 2: Related parallel decoding methods.

## 2 RELATED WORK

Existing research on parallelizing large language model inference can be categorized into two directions: enhancing response quality or accelerating generation speed. Quality-oriented methods (Brown et al., 2024; Wiseman & Rush, 2016; Wang et al., 2022; Chen et al., 2024; Zhang et al., 2024a; Yao et al., 2023) improve performance through multiple parallel sampling iterations but focus only on test-time scaling without optimizing sampling efficiency. Parallel scaling (Chen et al., 2025) enhances parallel computation in both training and inference by applying multiple learnable input transformations. In contrast, our work aligns with speed-oriented parallelization, which exploits inherent parallelism within a single response. This is achieved by decomposing sequential generation into parallelizable units while preserving textual coherence. A systematic taxonomy of this category is presented below.

**Orthogonal Acceleration Techniques** Speculative decoding, shown in Figure 2a, has recently emerged as a promising approach for accelerating LLM inference (Leviathan et al., 2023; Cai et al., 2023; Li et al., 2024a;b; 2025; Zhang et al., 2024b; Yi et al., 2024; Zhang et al., 2023). While effective, these techniques remain inherently sequential at the token level due to the autoregressive constraint, which fundamentally limits their achievable speedup.

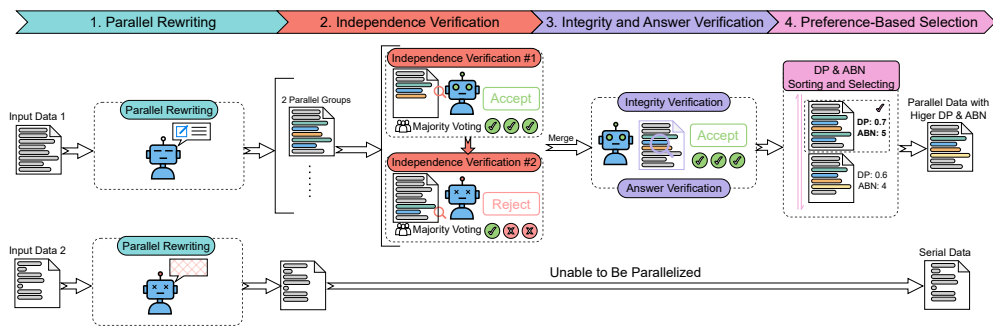
**Parallelization via Prompt Engineering** This category exploits prompt engineering to facilitate parallel generation. As shown in Figure 2a, SoT (Ning et al., 2023) employs a two-phase outline-and-expand strategy, which introduces considerable overhead due to KV-cache reinitialization, batch processing, and restrictive prompt templates. PDOS (Yu, 2025) mitigates some of these issues through internal masks and logits processors, yet as a prompt-based approach, it still underutilizes the model’s parallel capabilities and incurs efficiency loss from content re-prefilling during mode transitions.

**Architecture-Modified Parallelization** There have a few lines of work that modify attention mechanisms or training procedures to enable parallelism: (1) Visible Branch Architectures. Systems like GroupThink (Hsu et al., 2025) and Hogwild (Rodionov et al., 2025) allow inter-branch communication but suffer from backtracking costs (*e.g.*, when branches generate overlapping content). These methods focus more on branch collaboration rather than accelerating through independent parallel branch decoding. (2) Hidden Branch Architectures. As shown in Figure 2b, APAR (Liu et al., 2024),

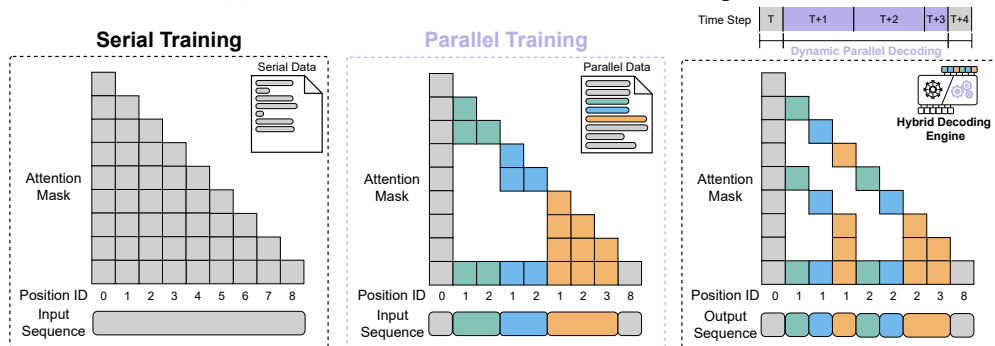
PASTA (Jin et al., 2025), and APR (Pan et al., 2025) enforce strict branch isolation but inevitably face key limitations. APAR discards parallel branch KV-caches during integration, which undermines contextual coherence; APR only exchanges abbreviated summaries across modes rather than full KV-states, leading to degraded continuity; and PASTA relies on pre-allocated positional ranges, causing encoding conflicts when actual branch lengths diverge from predictions. Moreover, APAR depends on predefined rules, while PASTA lacks further validation of independence and completeness across parallel branches, which ultimately restricts their ability to exploit intrinsic parallelism.

Our concurrent work Multiverse (Yang et al., 2025b), focusing on the Mathematical Reasoning task, also explores parallelism in large language model generation, implementing parallel branch generation through SGLang (Zheng et al., 2024) and leveraging Radix Attention for KV cache reuse. In contrast, our approach performs decoding within a single sequence, providing inherent continuity of the KV cache to directly reuse.

The proposed **ASPD** framework better harnesses intrinsic parallelism through a non-invasive data transformation pipeline. By introducing branch-invisible masks and shared position encodings, we enable lossless and seamless transitions between serial and parallel decoding modes. We systematically investigate diverse model architectures and inference paradigms, demonstrating significant acceleration effects without quality degradation across multiple domains, particularly in general tasks and retrieval-augmented generation scenarios.



(a) Non-Invasive Parallel Data Transformation Pipeline.



(b) Serial-parallel training details (left-center). Hybrid Decoding Engine (right).

Figure 3: An overview of our **Adaptive Serial-Parallel Decoding Framework**.

### 3 METHODOLOGY

Our framework unlocks the intrinsic parallel capabilities of auto-regressive (AR) large language models through two key aspects: mining inherent parallelism in AR model responses and adapting internal model architectures. We achieve these objectives through three key components below:

### 3.1 NON-INVASIVE PARALLEL DATA TRANSFORMATION PIPELINE

To effectively exploit this latent parallelism, we propose a novel non-invasive parallelization pipeline. Figure 3a illustrates the four foundational stages of our data curation pipeline. Given an original data sample  $\langle Q, A \rangle$ , where  $Q$  is the question and  $A$  is the model’s response, we proceed as follows:

1. **Parallel Rewriting.** This step transforms the serial response  $A$ , which may contain implicit parallel structures, into a format with explicit parallel structure markup. This is achieved by feeding  $A$  into the `Parallel Rewriting Prompt` and invoking an LLM. Specifically, multiple parallel branches (marked with the `<branch>` tag) are grouped into a parallel group (enclosed by the `<branchgroup>` tag), while serial content remains unchanged. The result is a rewritten answer, denoted as  $A_{pr}$ . The markup format is defined as:

```
<branchgroup>
<branch title="title 1">branch 1 content</branch>
<branch title="title 2">branch 2 content</branch>
...
</branchgroup>
```

To promote diversity, this process is repeated  $N$  times (with  $N = 3$  by default), yielding a set of candidate rewritten answers:  $\{A_{pr_1}, A_{pr_2}, \dots, A_{pr_N}\}$ .

2. **Independence Verification.** For each candidate  $A_{pr_j}$  from the previous step, we iterate over every `<branchgroup>` to verify the mutual independence of its branches. The `Independence Verification Prompt` is populated with the original question  $Q$ , the content of all branches in the group, and any preceding serial text. The LLM is then prompted to assess whether the branches can be processed independently. If the majority of LLM judgments indicate dependence, the group is **reverted to serial form**. If all parallel groups in  $A_{pr_j}$  fail the independence check, the candidate is discarded. Otherwise, the validated candidate—containing only verified parallel groups—is denoted as  $A_{iv_j}$  and proceeds to the next stage.
3. **Integrity and Answer Verification.** For each surviving candidate  $A_{iv_j}$ , we first generate a temporary serialized version  $A_{ia_j}$  by flattening all parallel groups back into sequential text. This  $A_{ia_j}$  is then evaluated against the original answer  $A$  using two separate prompts: the `Integrity Verification Prompt` checks for structural consistency, while the `Answer Verification Prompt` assesses answer correctness. A candidate  $A_{iv_j}$  is retained only if both checks pass under majority voting. Candidates that fail either verification are discarded.
4. **Preference-Based Selection.** Among the  $M$  candidates  $\{A_{iv_1}, A_{iv_2}, \dots, A_{iv_M}\}$  that pass all prior stages, we compute two metrics for each: the **Degree of Parallelism (DP)** and the **Average Branch Number (ABN)**. The candidate with the highest DP and ABN is selected as the final parallel-rewritten answer, denoted  $A_{iv}$ . The resulting parallel-structured data pair  $\langle Q, A_{iv} \rangle$  is first transformed into the format supported by the subsequent Hybrid Decoding Engine, and is then included in the curated dataset.

If all candidates fail verification in Steps 2 or 3, the pipeline terminates for this sample, and the original data  $\langle Q, A \rangle$  is preserved as the output. For each of the first three steps, input-output examples are provided in the corresponding prompts, all of which are shown in Appendix A.8.

### 3.2 INTERNAL PARALLELIZATION MODULE

#### Model Architecture for Native Parallelization

After obtaining the parallelized corpus described in Section 3.1, we need to modify the model architecture to enable efficient serial-parallel decoding. As shown in Figure 3b, we need to handle both the visibility between branches and the position encoding within it. To better leverage the capabilities of the native autoregressive model during parallel phases while supporting seamless transitions between parallel and sequential modes, we introduce two key components: (1) an internal parallel

mask for branch-independent parallel decoding, and (2) shared positional encodings across parallel branches at the same timestamp.

**Preliminaries** The generation process comprises a sequence of interleaved stages, each decoding in serial or parallel mode. In serial stages, generation proceeds through a single main branch in an autoregressive manner. During parallel stages, the model simultaneously decodes multiple parallel branches, enabling concurrent token generation across different aspects. Here,  $b(i)$  denotes the branch index of token  $i$  and  $t$  denotes temporal timestamps,  $stage\_start(i)$  denotes the starting position of the stage where token  $i$  is located, and  $\Delta i$  represents the relative position of token  $i$  within its stage.  $P_t$  represents the number of tokens being decoded simultaneously at time  $t$ .

$$\text{Attn}(Q, K, V) = \text{softmax} \left( \frac{QK^\top}{\sqrt{d_k}} + M \right) V \quad (1)$$

$$M_{i,j} = \begin{cases} 0 & \text{if } \mathcal{S}(b(i), b(j)) = 1 \text{ and } \text{pos}(i) > \text{pos}(j) \\ -\infty & \text{otherwise} \end{cases} \quad (2)$$

$$\mathcal{S}(b(i), b(j)) = \begin{cases} 1 & \text{if } b(i) \text{ in main branch} \\ 1 & \langle b(i), b(j) \rangle \text{ in same parallel branch} \\ 1 & \langle b(i), b(j) \rangle \text{ in different stage} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

$$\text{pos}(i) = \begin{cases} \sum_{t=1}^{i-1} P_t + 1 & \text{if } i \text{ in main branch} \\ stage\_start(i) + \Delta i & \text{if } i \text{ in para branch} \end{cases} \quad (4)$$

We extend the vanilla causal attention (Eq. 1) by incorporating a visibility function  $\mathcal{S}$ . As shown in Eq. 2,  $M_{i,j}$  is only visible when the visibility function  $\mathcal{S}$  equals 1 and position  $j$  precedes position  $i$ . Eq. 3 defines the visibility between tokens, where the main branch can see all other branches, while for parallel branches, visibility is enabled when  $\langle b(i), b(j) \rangle$  spans across stages or belongs to the same parallel branch. For position encoding, as shown in Eq. 4, tokens in the main branch maintain absolute positions in the flattened sequence, while parallel branches synchronize their position encodings at each timestamp.

### 3.3 HYBRID DECODING ENGINE

To enable parallelization, we augment the vocabulary with six special tokens before training: `<title>`, `</title>`, `<branch>`, `</branch>`, `<para>`, and `</para>`. LLMs will learn how to use these tokens in the training phase. During inference, when the model determines that the current response can be parallelized, it generates several parallel titles, each enclosed by `<title>` and `</title>`. After all branch titles are generated, the model outputs a `<para>` token, turning the engine into parallel decoding mode. For each parallel branch  $i$ , we set the prefix "`<branch>` $T_i$ :" to help the model identify which branch to generate, where  $T_i$  is the corresponding title for branch  $i$ . A branch is considered complete when the `</branch>` token is reached. Once all branches are finished, a `</para>` token is appended to the output sequence, switching the engine back to serial decoding mode. Subsequently, the following decoding process iteratively repeats the aforementioned parallel and serial steps, achieving adaptive serial-parallel decoding. As shown in Figure 3b, the hybrid engine implements the branch-independent parallel mask and synchronized position-id mechanism proposed in Section 3.2 to support parallel decoding. Each parallel token attends only to tokens from both the main branch and its respective branch, with sequentially increasing position-ids. This allows each branch to maintain the same generation pattern as native autoregressive models from its own perspective. After all parallel branches complete decoding, sequential decoding resumes on the main branch, where the position id of the first token reflects its actual position in the complete sequence. Our hybrid decoding engine supports iterative serial-parallel switching for optimal efficiency.



## 4 EXPERIMENTS

### 4.1 EXPERIMENTAL SETUP

**Dataset** Following APAR (Liu et al., 2024), we use ShareGPT Vicuna dataset (LMSYS, 2023) as our training data. This dataset encompasses instructions across various scenarios including STEM, roleplay, reasoning, and extraction tasks. Additionally, we utilize our internal Machine Reading Comprehension (MRC) dataset for multi-dimensional validation (Corresponding experimental results can be found in Appendix A.5). All data processing is conducted using our proposed non-invasive parallelization transformation pipeline to obtain parallelized data. For fair comparison, we create corresponding sequential data by removing special parallel tokens.

**Evaluation** We conduct comprehensive evaluations using established benchmarks including Vicuna Bench (Chiang et al., 2023) and MT Bench (Zheng et al., 2023), following the evaluation protocol of APAR (Liu et al., 2024). Furthermore, to evaluate models’ generalization capability, we introduce an out-of-domain benchmark, namely the RAG Bench. This benchmark consists of the first 200 questions with corresponding context sampled from rag-dataset-12000 (Neural Bridge AI, 2023).

Our evaluation framework encompasses two primary dimensions: **(1) Effectiveness Metrics.** We utilize the LLM-as-judge evaluation framework (Zheng et al., 2023) to quantify response quality, maintaining methodological consistency with APAR (Liu et al., 2024). All evaluations are conducted using Qwen3-235B-A22B (Yang et al., 2025a) to ensure balanced assessment. **(2) Efficiency Metrics.** We employ **Tokens-Per-Second (TPS)** as the primary throughput metric. For parallel models, we further introduce four additional metrics to characterize parallelization performance: **Parallel-Tokens-Per-Second (P-TPS, TPS in parallel stage)**, **Degree of Parallelism (DP, ratio of parallel to total tokens)**, **Average Branch Num (ABN, average parallel branch num)**, **Proportion of Parallel Data (PPD)**.

Table 1: Performance comparison on MT Bench and Vicuna Bench. The **bold** and underlined values denote the best and second-best results, respectively.

Method \ Bench	V-Ori	V-Seq	V-APAR	SoT	V-APAR*	V-ASPD	Q-Ori	Q-Seq	Q-ASPD
<b>MT Bench</b>	4.86	<b>5.59</b>	4.88	4.48	<u>5.38</u>	<b>5.59</b>	7.82	<u>7.98</u>	<b>8.15</b>
<b>Vicuna Bench</b>	6.21	<u>7.70</u>	6.10	5.93	7.62	<b>7.74</b>	8.65	<b>9.11</b>	<u>9.03</u>

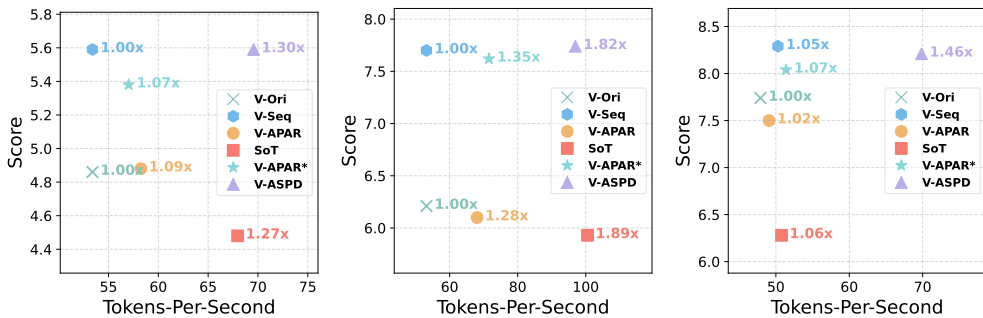


Figure 4: Speed-quality trade-off analysis across different parallel decoding methods on various benchmarks.

**Models** Following APAR (Liu et al., 2024), we employ Vicuna-V1.3-7B (Zheng et al., 2023) as our base model to ensure fair comparison across different approaches. To demonstrate the cross-architecture generalization capability of our method, we further conduct experiments using Qwen2.5-7b-Instruct (Qwen Team, 2024) as an additional foundation model.

**Implementation Details** During the training phase, we employed an initial learning rate of 1e-5 and a global batch size of 16 for three training epochs. The training process utilized cosine learning

rate scheduling with a warmup ratio of 0.1. The context length for training data was set to 8k tokens. Both sequential and parallel models used identical parameter configurations to ensure fair comparison. For inference, we use a batch size of 1, which aligns with common practices in parallel decoding literature, consistent with works such as PASTA (Jin et al., 2025) and Multiverse (Yang et al., 2025b). We consistently set the temperature parameter to 0.7, top\_k to 20, and top\_p to 0.8 across all experiments.

## 4.2 MAIN RESULTS

Tables 1 present a systematic evaluation across model architectures and methodologies. We evaluate our approach using two base models: Vicuna-1.3-7B (V) and Qwen2.5-7B-Instruct (Q). For each base model, we compare the original model (Ori), sequential fine-tuned model (Seq), SoT (Ning et al., 2023), APAR (Liu et al., 2024). Additionally, to ensure fair comparison, we utilize APAR’s official codebase and enhance its training data quality using Qwen3-235B-A22B to obtain APAR\*. The scores corresponding to each subtask are provided in Appendix A.2, and the efficiency metrics results are shown in Appendix A.2. The experimental results demonstrate the following:

**Superior Acceleration While Maintaining Quality** In Tables 1, both our fine-tuned parallel and serial models outperform the original model, establishing a solid foundation for our subsequent evaluations. Compared to the V-APAR and SoT method, our V-ASPD achieves a 14.55% and 24.78% improvement on the MT Bench, a 26.89% and 30.52% enhancement on the Vicuna Bench. In terms of decoding acceleration, presented in Figure 4, V-ASPD also achieves superior speed-quality balance across all benchmarks, with average acceleration ratios of 1.30x to 1.82x. Specifically, on Vicuna Bench, V-ASPD reaches 1.82x speedup, surpassing V-APAR (1.28x) and V-APAR\* (1.35x), while nearly matching SoT’s 1.89x that requires intensive memory usage. However, SoT’s quality score of 5.93 is notably lower than both V-ASPD (7.74) and baseline (6.21), limited by its rigid prompt design. Furthermore, we observe speedup ratios ranging from 1.0x to 3.10x across Coding to Writing, outperforming V-APAR, V-APAR\* and SoT on most subtasks (Detailed results are shown in Figure 6 in Appendix A.2). The consistent performance gains across diverse evaluation scenarios highlight the robustness and effectiveness of our parallel decoding framework.

**Cross-Domain and Cross-Model Generalization** As shown in Figure 4c, we observed that other competing methods face generalization issues on the out-of-domain RAG Bench. Especially, SoT’s speedup drops to 1.06x compared to V-ASPD’s 1.46x, due to requiring redundant prefilling of long context for each point during the point expanding stage. In contrast, our method not only achieves comparable generation quality to V-Seq but also maintains the highest speedup. The results presented in Table 1, which are based on the Qwen2.5-7B-Instruct model, highlight the consistent effectiveness of ASPD across model architectures. On MT Bench, Q-ASPD achieves a remarkable score of 8.15, demonstrating superior performance by surpassing both Q-Ori and Q-Seq models, with a notable 2.1% improvement compared to Q-Seq. Vicuna Bench shows comparable performance between Q-ASPD (9.03) and Q-Seq (9.11), with a mere 0.9% difference, confirming ASPD’s output quality preservation. The minimal performance variation across benchmarks underscores ASPD’s robustness.

## 4.3 PARALLELISM AT THE REASONING FRONTIER.

While previous approaches like APAR excluded mathematical and coding tasks from parallel data processing, we observe that our concurrent research work Multiverse (Yang et al., 2025b) has also directed attention toward mathematical problem parallelization. To thoroughly investigate this domain, we extended our model architecture to Qwen2.5-32B-Instruct (Qwen Team, 2024) and conducted comprehensive validation on these complex reasoning domains. Our evaluation includes a diverse range of benchmarks, including GPQA (Rein et al., 2024), MATH500 (Hendrycks et al., 2021), and competition-level mathematics such as AMC23 (MAA, 2023), AIME24 (MAA, 2024), and AIME25 (MAA, 2025). For training data, we utilize OpenR1-Math-220K (Hugging Face, 2025), which was generated by prompting DeepSeek-R1 (Guo et al., 2025) with NuminaMath1.5 (LI et al., 2024) as the query set. The model was trained for 9 epochs with a global batch size of 88, using a 12k context window and a learning rate of 1e-5.

As demonstrated in Table 2 (reporting pass@1 scores based on Evalchemy (Raouf et al., 2025), with AMC and AIME results representing means across 8 random seeds), ASPD exhibits performance



Table 2: Math bench performance with Qwen2.5-32B-Instruct.

Model \ Bench	Ori	Seq	ASPD
<b>MATH500</b>	82.00	<b>94.40</b>	<u>94.00</u>
<b>AMC23</b>	62.19	<b>89.69</b>	<u>89.38</u>
<b>GPQA</b>	48.99	<u>61.11</u>	<b>65.66</b>
<b>AIME2024</b>	17.50	<u>58.75</u>	<b>62.08</b>
<b>AIME2025</b>	12.50	<u>47.92</u>	<b>50.00</b>

Table 3: Comparative analysis of parallelization metrics on mathematical benchmarks between **ASPD** and **Seq**.

Metric \ Bench	PPD	DP	ABN	TPS	P-TPS
<b>MATH500</b>	88.40	33.30	2.61	27.14 <sub>1.17x</sub>	43.03 <sub>1.86x</sub>
<b>AMC23</b>	84.38	22.24	2.80	21.93 <sub>1.11x</sub>	39.30 <sub>1.99x</sub>
<b>GPQA</b>	66.16	32.70	2.88	22.06 <sub>1.13x</sub>	36.57 <sub>1.88x</sub>
<b>AIME2024</b>	65.42	8.84	2.48	16.43 <sub>1.04x</sub>	24.37 <sub>1.54x</sub>
<b>AIME2025</b>	79.17	8.60	2.40	15.77 <sub>1.08x</sub>	26.82 <sub>1.83x</sub>

gains of 12%, 27.19%, 16.67%, 44.58%, 37.5% over **Ori** across MATH500, AMC23, GPQA, AIME24 and AIME25 respectively. Notably, ASPD achieves acceleration ratios of 1.04-1.17 in TPS and 1.54-1.99 in P-TPS versus the **Seq** model, while maintaining performance within a range of -0.4% to +5%, demonstrating robust effectiveness.

#### 4.4 ABLATION STUDY

##### 4.4.1 IMPACT OF DATA PROCESSING PIPELINE.

While autoregressive models inherently possess parallelizable characteristics in their responses, precisely identifying parallelizable components remains a significant challenge. Ablation of data pipeline in Table 4 demonstrates the performance variations across different data processing methodologies. APAR achieves only 1.11x speedup compared to the baseline, due to its rule-based parallelization approach which fails to trigger parallel responses in many scenarios. PASTA exhibits the lowest performance score of 4.98, primarily because its data processing pipeline lacks consideration for branch independence verification. In contrast, our method achieves optimal results in both effectiveness (score of 7.64) and efficiency (TPS of 104.21), demonstrating that our Non-Invasive data transformation pipeline can better leverage the model’s inherent parallel capabilities while maintaining superior response quality.

Table 4: Ablation studies on data pipeline efficiency, attention mask strategies, and position encoding schemes for parallel decoding. \* denotes implementation with official codebase, † denotes implementation with official prompt.

Method	Data Pipeline		Attention Mask				Position Id		
	Score	TPS	PosId	Attn	Score	TPS	PosId	Score	TPS
Baseline	<u>6.21</u>	53.19	Seq	Shared	4.64	<b>110.30</b>	Predict	<u>6.75</u>	<u>72.15</u>
APAR*	5.81	59.25	<b>Seq</b>	<b>Indep</b>	<b>7.64</b>	<u>104.21</u>	Same-Max	6.78	89.45
PASTA†	4.98	<b>106.83</b>	Max	Shared	3.70	86.96	Same-Re	<u>7.29</u>	<u>95.24</u>
<b>ASPD</b>	<b>7.64</b>	<u>104.21</u>	Max	Indep	<u>6.78</u>	89.45	<b>Same-Seq</b>	<b>7.64</b>	<b>104.21</b>

##### 4.4.2 IMPACT OF MASK VISIBILITY

While GroupThink (Hsu et al., 2025) and Hogwild (Rodionov et al., 2025) have explored visible masks for agent collaboration, our approach differs fundamentally in its objective. As demonstrated in attention mask ablation of Table 4, we define two mask types: **Shared** and **Indep** (Independent), where branches can or cannot see each other respectively. We also introduce two types of position id setting: **Max** (uses maximum position id across parallel branches for merging) and **Seq** (assigns actual sequential position id to the first token of the merged main branch, as illustrated in Figure 3b). Our empirical evaluation shows that *Shared* masks consistently outperform *Indep* masks across both *Seq* and *Max* position id configurations. This architectural choice is motivated by our primary objective of decomposing responses into independent, self-contained parallel branches to achieve optimal computational efficiency. This empirical finding strongly validates our design decision to maintain strict branch isolation as an optimal strategy for parallel response generation.

#### 4.4.3 POSITION ENCODING PARADIGMS

During parallel decoding, multiple branches decode simultaneously while remaining mutually invisible, making the arrangement of position ids particularly crucial. Based on whether position ids are consistent across parallel branches, we abstract position encoding schemes into the following two categories: (1) **Predict**. Following PASTA (Jin et al., 2025), we allocate position ids to each parallel branch by predicting its length. Before parallel decoding starts, the model predict the prospective length of every branch and assigns initial position ids accordingly. We adopt PASTA’s best-performing configuration, *Predict-10X*, which multiplies the predicted length by ten to set the decoding horizon for each branch. (2) **Same**. The following three variants use identical position ids across parallel branches at each timestamp, differing in their strategies for reordering position ids when merging parallel branches into the main branch. *Same-Max* equals to **Max** in Section 4.4.2. *Same-Re* (Same-Rearrange) reorders position ids sequentially between parallel branches. *Same-Seq* (Same-Sequential) equals to **Seq** in Section 4.4.2.

As evidenced in position id ablation of Table 4, [due to inconsistencies between actual branch lengths during decoding and predicted lengths, the \*Predict\* strategy yields the poorest performance](#). Among the *Same* family approaches, our adopted *Same-Seq* strategy emerges as the most effective strategy, achieving a Score of 7.64 and TPS of 104.21, outperforming both *Predict*, *Same-Re* and *Same-Max* variants in terms of quality and efficiency. This suggests that maintaining natural position ordering while sharing timestamps across parallel branches optimally preserves the model’s positional understanding.

## 5 CONCLUSION

In this work, we present **ASPD: The Adaptive Serial-Parallel Decoding** framework for efficient hybrid decoding within auto regressive large language models. The proposed method introduces a non-invasive parallel data transformation pipeline and internal parallelization with branch-invisible attention masks and shared position ids, achieving substantial latency reduction while maintaining response quality compared to traditional autoregressive LLMs. Our extensive experiments demonstrate state-of-the-art performance across various benchmarks including general tasks, retrieval-augmented generation, and mathematical reasoning. Furthermore, we establish a novel paradigm for parallel decoding that eliminates external overheads from batching, threading or re-prefill between serial-parallel switching, providing valuable insights for future research in efficient LLM inference. The strong empirical results and theoretical contributions of ASPD suggest promising applications in latency-critical scenarios.

## REPRODUCIBILITY STATEMENT

All experimental results presented in this work are fully reproducible. We provide comprehensive experimental settings in Section 4.1, including detailed implementation specifications covering datasets, evaluation protocols, and model architectures. To facilitate complete reproducibility of our research findings, we have made available a comprehensive anonymous code repository (referenced in the abstract) containing three essential components:

1. A parallel data construction pipeline (located in the `data_ppl` directory), which implements our Non-Invasive Parallel Data Transformation Pipeline.
2. A training framework implemented for the Internal Parallelization Module (located in the `train` directory), supporting for serial-parallel training.
3. An adaptive hybrid inference engine (located in the `infer` directory), which enables efficient serial-parallel decoding.

Each component is accompanied by thorough documentation and step-by-step operational guidelines to ensure complete reproducibility of our results. In accordance with the double-blind review process, all materials have been carefully anonymized to maintain review integrity.

## REFERENCES

- 540  
541  
542 Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and  
543 Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling.  
544 *arXiv preprint arXiv:2407.21787*, 2024.
- 545 Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, and Tri Dao. Medusa: simple frame-  
546 work for accelerating llm generation with multiple decoding heads (2023). URL <https://github.com/FasterDecoding/Medusa>. *GitHub repository*, 2023.
- 548 Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei A Zaharia, and  
549 James Y Zou. Are more llm calls all you need? towards the scaling properties of compound ai  
550 systems. *Advances in Neural Information Processing Systems*, 37:45767–45790, 2024.
- 552 Mouxiang Chen, Binyuan Hui, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Jianling Sun, Junyang Lin, and  
553 Zhongxin Liu. Parallel scaling law for language models. *arXiv preprint arXiv:2505.10475*, 2025.
- 554 Wei-Lin Chiang, Zhuohan Li, Ziqing Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng,  
555 Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot  
556 impressing gpt-4 with 90%\* chatgpt quality. See <https://vicuna.lmsys.org> (accessed 14 April  
557 2023), 2(3):6, 2023.
- 559 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,  
560 Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms  
561 via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- 562 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song,  
563 and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv  
564 preprint arXiv:2103.03874*, 2021.
- 565 Chan-Jan Hsu, Davide Buffelli, Jamie McGowan, Feng-Ting Liao, Yi-Chang Chen, Sattar Vakili,  
566 and Da-shan Shiu. Group think: Multiple concurrent reasoning agents collaborating at token level  
567 granularity. *arXiv preprint arXiv:2505.11107*, 2025.
- 568 Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL <https://github.com/huggingface/open-r1>.
- 571 Tian Jin, Ellie Y Cheng, Zack Anker, Nikunj Saunshi, Blake M Elias, Amir Yazdanbakhsh,  
572 Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. Learning to keep a promise:  
573 Scaling language model decoding parallelism with learned asynchronous decoding. *arXiv  
574 preprint arXiv:2502.11517*, 2025.
- 575 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E.  
576 Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model  
577 serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating  
578 Systems Principles*, 2023.
- 580 Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative  
581 decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.
- 582 Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang,  
583 Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann  
584 Fleureau, Guillaume Lample, and Stanislas Polu. NuminaMath. [<https://huggingface.co/AI-MO/NuminaMath-1.5>] ([https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina\\_dataset.pdf](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf)), 2024.
- 587 Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires  
588 rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024a.
- 589 Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language  
590 models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 2024b.
- 592 Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceler-  
593 ation of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025.

- 594 Mingdao Liu, Aohan Zeng, Bowen Wang, Peng Zhang, Jie Tang, and Yuxiao Dong. Apar: Llms can  
595 do auto-parallel auto-regressive decoding. *arXiv preprint arXiv:2401.06761*, 2024.
- 596
- 597 LMSYS. Sharegpt vicuna unfiltered, 2023. URL [https://huggingface.co/datasets/  
598 anon8231489123/ShareGPT\\_Vicuna\\_unfiltered](https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered).
- 599 MAA. American mathematics competitions. In *American Mathematics Competitions*, 2023.
- 600
- 601 MAA. American invitational mathematics examination 2024, 2024. URL [https://  
602 //artofproblemsolving.com/wiki/index.php/American\\_Invitational\\_  
603 Mathematics\\_Examination](https://artofproblemsolving.com/wiki/index.php/American_Invitational_Mathematics_Examination). Accessed: 2025-05-14.
- 604 MAA. American Invitational Mathematics Examination 2025, 2025. URL [https://  
605 //artofproblemsolving.com/wiki/index.php/American\\_Invitational\\_  
606 Mathematics\\_Examination](https://artofproblemsolving.com/wiki/index.php/American_Invitational_Mathematics_Examination). Accessed: 2025-05-14.
- 607
- 608 Neural Bridge AI. Retrieval-augmented generation (rag) dataset 12000, 2023. URL [https://  
609 huggingface.co/datasets/neural-bridge/rag-dataset-12000](https://huggingface.co/datasets/neural-bridge/rag-dataset-12000).
- 610 Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-  
611 thought: Prompting llms for efficient parallel generation. *arXiv preprint arXiv:2307.15337*, 2023.
- 612
- 613 Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer,  
614 and Alane Suhr. Learning adaptive parallel reasoning with language models. *arXiv preprint arXiv:  
615 2504.15466*, 2025.
- 616 Qwen Team. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.
- 617
- 618 Negin Raouf, Etash Kumar Guha, Ryan Marten, Jean Mercat, Eric Frankel, Sedrick Keh, Hritik  
619 Bansal, Georgios Smyrnis, Marianna Nezhurina, Trung Vu, Zayne Rea Sprague, Mike A  
620 Merrill, Liangyu Chen, Caroline Choi, Zaid Khan, Sachin Grover, Benjamin Feuer, Ashima  
621 Suvarna, Shiye Su, Wanxia Zhao, Kartik Sharma, Charlie Cheng-Jie Ji, Kushal Arora, Jeffrey  
622 Li, Aaron Gokaslan, Sarah M Pratt, Niklas Muennighoff, Jon Saad-Falcon, John Yang, Asad  
623 Aali, Shreyas Pimpalgaonkar, Alon Albalak, Achal Dave, Hadi Pouransari, Greg Durrett, Se-  
624 woong Oh, Tatsunori Hashimoto, Vaishaal Shankar, Yejin Choi, Mohit Bansal, Chinmay Hegde,  
625 Reinhard Heckel, Jenia Jitsev, Maheswaran Sathiamoorthy, Alex Dimakis, and Ludwig Schmidt.  
Evalchemy, June 2025. Software.
- 626
- 627 David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Di-  
628 rani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a bench-  
629 mark. In *First Conference on Language Modeling*, 2024.
- 630
- 631 Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazar-  
632 ian, Anton Sinitin, Denis Kuznedelev, and Dan Alistarh. Hogwild! inference: Parallel llm  
generation via concurrent attention. *arXiv preprint arXiv:2504.06261*, 2025.
- 633
- 634 ByteDance Seed, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang,  
635 Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, et al. Seed1. 5-thinking: Advancing superb rea-  
636 soning models with reinforcement learning. *arXiv preprint arXiv:2504.13914*, 2025.
- 637
- 638 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdh-  
639 ury, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models.  
*arXiv preprint arXiv:2203.11171*, 2022.
- 640
- 641 Sam Wiseman and Alexander M Rush. Sequence-to-sequence learning as beam-search optimization.  
*arXiv preprint arXiv:1606.02960*, 2016.
- 642
- 643 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu,  
644 Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint  
645 arXiv:2505.09388*, 2025a.
- 646
- 647 Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. Multiverse: Your language mod-  
els secretly decide how to parallelize and merge generation. *arXiv preprint arXiv:2506.09991*,  
2025b.

- 648 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik  
649 Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Ad-  
650 vances in neural information processing systems*, 36:11809–11822, 2023.
- 651 Hanling Yi, Feng Lin, Hongbin Li, Peiyang Ning, Xiaotian Yu, and Rong Xiao. Generation meets  
652 verification: Accelerating large language model inference with smart parallel auto-correct decod-  
653 ing. *arXiv preprint arXiv:2402.11809*, 2024.
- 654 Yijiong Yu. Accelerate parallelizable reasoning via parallel decoding within one sequence. *arXiv  
655 preprint arXiv:2503.20533*, 2025.
- 656 Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. Accessing gpt-4 level  
657 mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint  
658 arXiv:2406.07394*, 2024a.
- 659 Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft &  
660 verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint  
661 arXiv:2309.08168*, 2023.
- 662 Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations  
663 for speculative sampling. *arXiv preprint arXiv:2408.15766*, 2024b.
- 664 Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang,  
665 Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and  
666 chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.
- 667 Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi  
668 Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of  
669 structured language model programs. *Advances in neural information processing systems*, 37:  
670 62557–62583, 2024.

## 675 A APPENDIX

### 676 A.1 THE USE OF LARGE LANGUAGE MODELS (LLMs)

677 During the drafting of this work, AI tools were employed solely for two purposes: (1) conducting  
678 grammar and syntax checks to ensure linguistic accuracy and consistency; (2) polishing partial de-  
679 scriptive content and result presentation. Notably, the core content of this work—including research  
680 design, methodology, analytical frameworks, key parameters, and original conclusions—was inde-  
681 pendently conceived, developed, and finalized by the authors. AI tools did not participate in any  
682 substantive content creation, decision-making, or scientific reasoning processes. All AI-assisted re-  
683 visions were further reviewed, verified, and adjusted by the authors to guarantee the academic rigor,  
684 authenticity, and integrity of the research content.

### 685 A.2 FULL PERFORMANCE COMPARISON ON MT BENCH AND VICUNA BENCH

686 Table 6 shows complete experimental results of various subtasks on MT Bench and Vicuna Bench,  
687 and the corresponding acceleration ratio of each subtask is presented in Figure 6.

### 688 A.3 WALL CLOCK LATENCY SPEEDUP

689 To evaluate the reduction in actual system overheads achieved by our method, we perform wall  
690 clock latency (WCL) speedup evaluations across benchmarks, and the results are consistent with the  
691 tokens/sec metrics we previously reported. As shown in Table 5, ASPD achieves the highest WCL  
692 speedup (44.90% on MT Bench, 33.90% on Vicuna Bench, and 37.64% on RAG Bench) compared  
693 to the sequential baseline (V-Seq) and the parallel baseline (V-APAR\*). Notably, ASPD maintains  
694 generation quality comparable to V-Seq (e.g., 5.59 vs. 5.59 on MT Bench) while delivering superior  
695

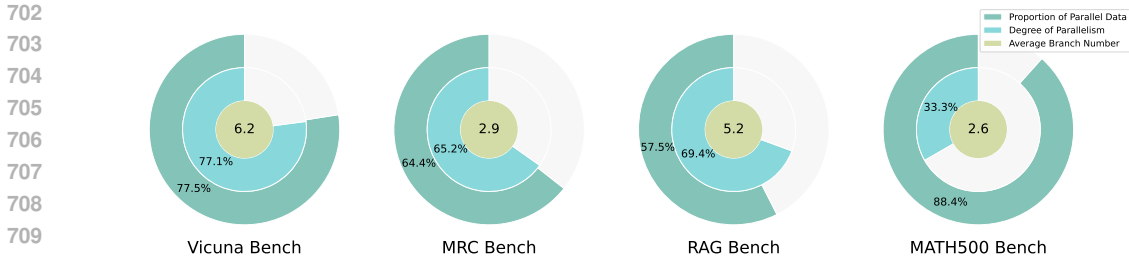


Figure 5: Parallelization patterns in our trained parallel model’s responses across different scenarios.

Table 5: Comparison of WCL (wall clock latency) speedup and generation quality. \* denotes implementation with official codebase.

Model	MT Bench		Vicuna Bench		RAG Bench	
	WCL (Speedup)	Score	WCL (Speedup)	Score	WCL (Speedup)	Score
V-Seq	15.5	5.59	17.64	7.70	9.67	8.29
V-APAR*	10.37 (33.10%)	5.38	12.42 (29.59%)	7.62	7.41 (23.37%)	8.04
V-ASPD	8.54 (44.90%)	5.59	11.66 (33.90%)	7.74	6.03 (37.64%)	8.21

speedup—further confirming the effectiveness of our method in balancing latency reduction and output quality.

#### A.4 DATA INTRINSIC PARALLELISM OF BENCHMARKS

Building on our analysis of parallelization potential (Figure 1), our method effectively leverages intrinsic parallelism while maintaining generalizability across diverse scenarios (Figure 5). Results on Vicuna (Chiang et al., 2023), our Internal MRC, and RAG benchmark demonstrate optimal parallelism utilization. The Mathematical Reasoning (MATH500) (Hendrycks et al., 2021) benchmark exhibits the highest parallelization constraints among evaluated tasks, with 88.4% PPD (+4.2%), 33.3% DP (+2.7%), and 2.6 ABN (-0.1), which exhibits strong alignment with the statistical patterns shown in Figure 1. These quantitative metrics provide compelling evidence that **our approach successfully harnesses the inherent parallelization capabilities of LLMs while preserving the fundamental characteristics of native autoregressive generation.**

#### A.5 GENERALIZATION ACROSS DIFFERENT BENCHMARKS

To evaluate the generalization capability of parallel processing across different benchmarks, we conducted training on our internal MRC benchmark and tested generalization performance on MRC, LR, and AI Search benchmarks, using Qwen2.5-7B-Instruct as our base model. Specifically, MRC is designed to evaluate a wide range of reading comprehension skills, including various question types such as single passage understanding, multi-information extraction, refusal to answer, etc. Logical Reasoning (LR) contains problems requiring step-by-step deductive reasoning, while AI Search resembles RAG scenarios where responses are generated by analyzing multi-source search results, typically presented in an itemized format. For these benchmarks, we implement human evaluation protocols to mitigate potential model-based assessment biases.

As shown in Table 8, on the homogeneous MRC benchmark, ASPD achieves a 1.35x speedup in TPS compared to the original model, while maintaining performance within 1% of the fine-tuned sequential model. Cross-validation using LR and AI Search benchmarks demonstrates acceleration ratios of 1.15x and 1.45x respectively.

For the AI Search test set, which contains more questions requiring itemized responses, we observe increased opportunities for parallel output generation. This leads to improved parallelization metrics including PPD, DP, and ABN (3.21 in AI Search versus 2.40 in Logical Reasoning), as illustrated in Table 7. Consequently, both TPS and P-TPS metrics show significant improvements. These results indicate that the effectiveness of our parallelization approach varies depending on the specific task characteristics.



Table 6: Performance comparison on MT Bench and Vicuna Bench. CS is short for Common-Sense on Vicuna Bench.

MT Bench									
Model \ Task	V-Ori	V-Seq	V-APAR	SoT	V-APAR*	V-ASPD	Q-Ori	Q-Seq	Q-ASPD
<b>Coding</b>	<b>3.50</b>	<u>3.10</u>	2.70	3.00	2.55	2.90	<b>7.15</b>	6.60	<u>6.70</u>
<b>Extraction</b>	<u>5.10</u>	<u>4.40</u>	<b>5.20</b>	3.08	5.05	4.80	<u>7.45</u>	6.93	<b>7.80</b>
<b>Humanities</b>	6.55	<b>8.40</b>	6.40	6.30	<u>8.10</u>	7.95	8.70	<b>9.15</b>	<u>9.05</u>
<b>Math</b>	2.95	<b>3.05</b>	2.55	2.75	2.50	<u>3.00</u>	<u>8.50</u>	8.45	<b>8.90</b>
<b>Reasoning</b>	4.65	4.80	<u>5.45</u>	4.50	4.75	<b>5.85</b>	6.85	<b>7.25</b>	<u>7.05</u>
<b>Roleplay</b>	5.30	<u>7.10</u>	5.60	5.60	6.80	<b>7.25</b>	8.10	<u>8.40</u>	<b>8.48</b>
<b>Stem</b>	5.55	6.20	5.55	5.85	<b>6.50</b>	<u>6.30</u>	7.80	<b>8.60</b>	<u>8.50</u>
<b>Writing</b>	5.30	<b>7.65</b>	5.60	4.75	<u>6.80</u>	<u>6.70</u>	8.00	<u>8.50</u>	<b>8.75</b>
<b>Mean</b>	4.86	<b>5.59</b>	4.88	4.48	<u>5.38</u>	<b>5.59</b>	7.82	<u>7.98</u>	<b>8.15</b>

Vicuna Bench									
Model \ Task	V-Ori	V-Seq	V-APAR	SoT	V-APAR*	V-ASPD	Q-Ori	Q-Seq	Q-ASPD
<b>Coding</b>	<b>4.29</b>	3.71	<u>4.14</u>	3.43	3.71	3.71	<b>9.00</b>	8.29	<u>8.71</u>
<b>CS</b>	7.40	<b>9.00</b>	7.20	7.30	<u>8.90</u>	<u>8.90</u>	<u>9.00</u>	<b>9.30</b>	<u>9.00</u>
<b>Counterfactual</b>	5.10	<u>8.50</u>	5.30	5.30	<u>8.40</u>	<b>8.60</b>	<u>8.40</u>	<b>9.10</b>	<b>9.10</b>
<b>Fermi</b>	<b>5.50</b>	4.50	4.60	4.40	5.00	<u>5.20</u>	7.70	<b>8.20</b>	<u>8.10</u>
<b>Generic</b>	7.30	<b>9.20</b>	7.40	7.40	<u>8.95</u>	8.80	8.70	<b>9.55</b>	<u>9.40</u>
<b>Knowledge</b>	7.30	<b>9.10</b>	7.20	7.50	8.60	<u>9.00</u>	9.00	<b>9.40</b>	<u>9.10</u>
<b>Math</b>	<u>2.67</u>	2.33	<b>3.33</b>	2.00	2.33	<u>2.67</u>	<u>9.67</u>	<b>10.00</b>	<b>10.00</b>
<b>Roleplay</b>	7.20	<b>9.20</b>	6.60	6.70	<u>9.00</u>	<u>9.00</u>	9.00	<u>9.30</u>	<b>9.40</b>
<b>Writing</b>	6.10	<u>8.80</u>	6.60	5.80	<u>8.80</u>	<b>9.00</b>	8.20	<b>9.20</b>	<u>9.00</u>
<b>Mean</b>	6.21	<u>7.70</u>	6.10	5.93	7.62	<b>7.74</b>	8.65	<b>9.11</b>	<u>9.03</u>

Table 7: Comprehensive parallelization metrics on our internal test sets.

Test Set	PPD	DP	ABN	P-TPS
MRC	64.40	65.21	2.86	109.32 <sub>2.18x</sub>
LR	44.59	55.80	2.40	101.72 <sub>1.92x</sub>
AI Search	64.50	67.69	3.21	97.27 <sub>2.41x</sub>

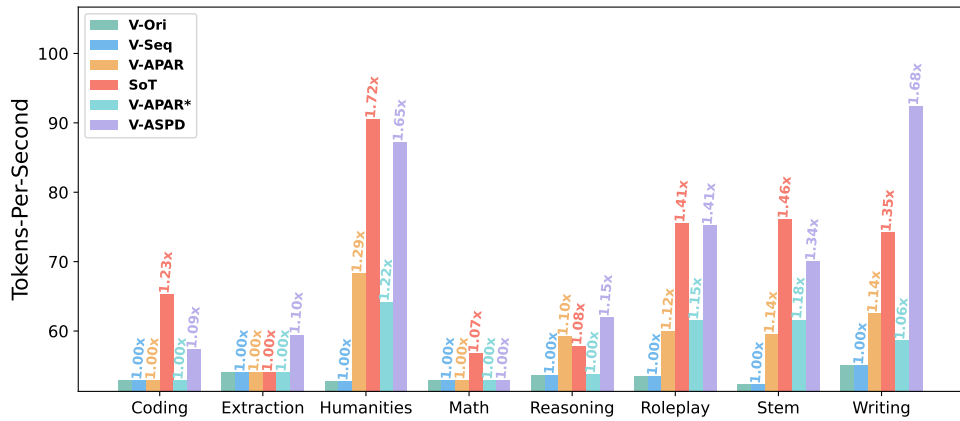
Table 8: Human evaluation on our internal test sets.

Model	MRC		LR		AI Search	
	ACC	TPS	ACC	TPS	Score	TPS
Q-Ori	64.40	50.18	53.50	53.06	7.85	40.29
Q-Seq	<b>73.20</b>	<u>50.53</u>	<b>56.69</b>	<u>54.32</u>	<b>8.23</b>	39.89
<b>Q-ASPD</b>	<u>72.40</u>	<b>67.91</b>	<u>55.41</u>	<b>61.26</b>	<u>8.15</u>	<b>58.56</b>

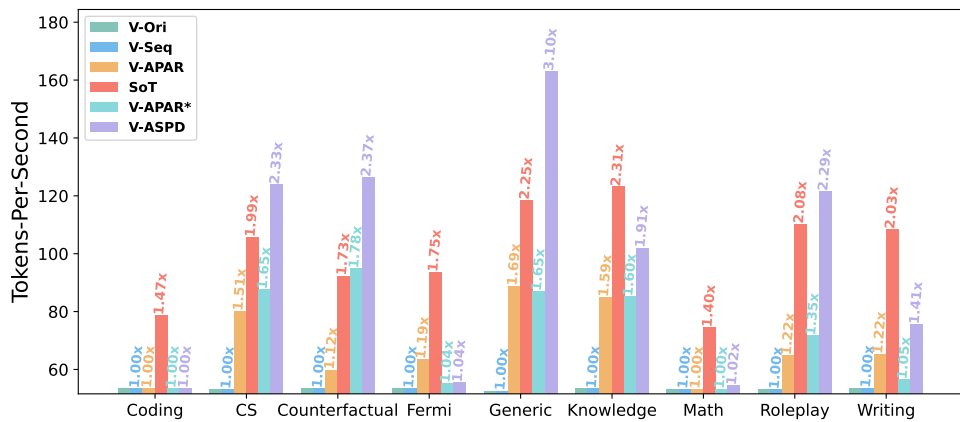
## A.6 IN-DEPTH ANALYSIS OF MATHEMATICAL RESULTS

**Parallelism Variation Across Tasks** Compared to the results of other scenarios in Figure 4, mathematical tasks demonstrate relatively lower acceleration benefits (1.17x speedup in MATH500, 1.46x in RAG, and 1.82x in Vicuna). In mathematical reasoning tasks, our parallel model contains reasoning processes with step-by-step deductions. These reasoning chains and strong inter-step dependencies lead to reduced DP, which aligns with the parallel pattern shown in Figure 1 (30.6% for mathematics versus approximately 68% for other domains). This reduction in parallelizable content consequently results in decreased TPS acceleration ratios.

**Parallelism Variation Across Difficulty Levels** As shown in Table 3, the DP significantly decreases from 33.30% in MATH500 to 8.6% in AIME2025, with TPS decreasing from 1.17x to 1.08x. This decline in parallelism indicates a strong correlation between performance degradation and difficulty level. Our detailed analysis of MATH500 in Figure 7a clearly shows that parallel efficiency metrics decrease as problem difficulty increases. This inverse relationship is consistently observed across various mathematical benchmarks in Figure 7b, where higher difficulty levels corresponds to

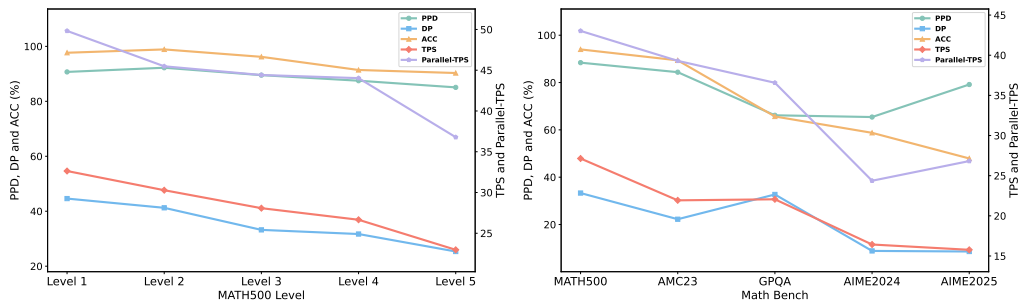


(a) MT Bench.



(b) Vicuna Bench.

Figure 6: Token Speed Analysis on Different Benchmarks. Our method achieves the best speedup performance on MT Bench and Vicuna Bench. For specific tasks, the speedup ratios reach 1.68-3.10x in scenarios like Generic, Writing, and Common-Sense (CS).



(a) Parallelization metrics vs. problem difficulty.

(b) Parallelization metrics across math benches.

Figure 7: Analysis between Parallelization Metrics and Problem Difficulty.

lower DP and TPS values. Notably, the GPQA benchmark presents an interesting exception - its multiple-choice format naturally enables more parallel processing opportunities, leading to higher parallelization metrics despite comparable difficulty levels with other benchmarks.

## 864 A.7 LIMITATIONS AND FUTURE WORK

865  
866 Our approach focuses on semantic-level parallelism through concurrent generation of independent  
867 response segments, while speculative decoding achieves token-level parallelism via predictive token  
868 generation with sequential verification. These orthogonal yet complementary approaches present  
869 opportunities for future research to combine both paradigms for enhanced acceleration.

870 Moreover, our parallelization modifications exhibit significant potential for seamless integration  
871 with mainstream inference frameworks, including vLLM (Kwon et al., 2023) and SGLang (Zheng  
872 et al., 2024), promising substantial enhancements in acceleration performance.

873 Within our parallel data production pipeline, we have established several key evaluation metrics,  
874 including TPS, DP, and ABN. These quantitative indicators demonstrate significant potential for  
875 integration into reinforcement learning frameworks, where they could effectively guide model op-  
876 timization towards achieving enhanced parallelism while maintaining or improving overall perfor-  
877 mance.

## 879 A.8 PROMPT TEMPLATES

880 We provide all prompt templates used in our Non-Invasive Parallel Data Transformation Pipeline.

### 883 Parallel Rewriting

884  
885 Your task is to rewrite the raw output text into parallel structures if necessary. If certain parts  
886 of the text are suitable for conversion into parallel format (where some content can be divided into  
887 multiple parallel blocks that are logically independent and don't depend on each other), this allows  
888 the model to generate these contents in parallel rather than sequentially, significantly improving  
889 decoding efficiency and response time. The parallel decoding process roughly follows these steps:

- 888 1. Serial content
- 889 2. Generate parallel content (can have multiple parallel branches)
- 890 3. Wait for all parallel branches to complete before continuing with serial content
- 891 4. Repeat steps 1-3 until output is complete

892 The core idea of parallel rewriting is: splitting the original output text into different parallel  
893 branches if possible, with these branches being independently completable without relying on each  
894 other's results. After these parallel branches are completed, the results are aggregated before  
895 proceeding with further parallel branch planning until the problem is solved. This parallel branch  
896 division can be applied to various tasks like reading comprehension or mathematics.

897 Note the input/output format:

895 Input format:  
896 [[Original Output]]  
897 [INPUT\_START]  
898 XXXX (raw output text of language model)  
899 [INPUT\_END]

899 Output format:  
900 [[Parallel Text]]  
901 [PARALLEL]  
902 X (X=true if parallel content exists, otherwise X=false)  
903 [PARALLEL]  
904 [OUTPUT\_START]  
905 XXXX (output rewritten parallel text if parallel content exists, otherwise nothing)  
906 [OUTPUT\_END]

907 First, you need to determine whether the given raw output text can be generated in parallel. If no  
908 parallel content exists, output false between [PARALLEL]s and nothing between [OUTPUT\_START] and  
909 [OUTPUT\_END]. If parallelization is possible, you need to annotate the language model output text  
910 with specific tags to highlight paragraphs suitable for parallel generation. Two parallel tags are  
911 used: <branch> and <branchgroup>, with the following instructions:

- 912 1. Use <branchgroup> tags to wrap portions of text that can be generated in parallel (parallel  
913 content), which can be divided by semantics, headings, steps, etc. The divided parallel branch  
914 content should be wrapped and distinguished by <branch> tags.
- 915 2. Use <branch> tags to divide <branchgroup> content, with each division representing a parallel  
916 branch. Each parallel branch can be generated asynchronously without depending on other branches'  
917 content. Therefore, <branch> tags shouldn't be applied to content that depends on current parallel  
918 branches as prerequisites. For each <branch> tag, generate a very concise title description as the  
919 tag's title attribute (format: <branch title="title word">). This title description can be used after  
920 closing async tags to ensure continuity and consistency, while ensuring that under the same  
921 <branchgroup>, different <branch> tags' title words aren't repeated (as this would prevent parallel  
922 decoding from distinguishing subsequent parallel content).
- 923 3. Within the same <branchgroup>, there shouldn't be any explicit or implicit dependencies between  
924 parallel branches that rely on other branches' intermediate results or conclusions - such parallel  
925 branch divisions would be unreasonable and need to be redivided appropriately. Additionally, if any  
926 branch depends on other branches as prerequisites, the parallel division is also unreasonable.
- 927 4. Use a single </branchgroup> closing tag for synchronization. All content generated before  
928 </branchgroup>, including text marked with <branch>, can be accessed after the </branchgroup> tag for  
929 subsequent text generation, ensuring continuity and consistency. Below are detailed specifications  
930 for tag annotation rules:

918  
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  
968  
969  
970  
971

Parallel Tag Annotation Rules:

- <branch>, </branch> tags and <branchgroup>, </branchgroup> tags are used in pairs.
- Between </branch> closing and <branch> opening tags, there should be no non-empty content (all parallel branch content must be within <branch> tags).
- Ensure each <branch> tag within an <branchgroup> is mutually independent, not relying on any other <branch> tag's results.
- Ensure each <branch> tag's content length exceeds its title attribute's length, with content being at least 10 characters (this maximizes parallel generation efficiency).
- <branch> tag content must come from the original output text and can be slightly adjusted for parallel branch requirements, but shouldn't modify original semantics or format. <branch> tag content should maintain the original output's format so that removing parallel tags restores the original output format.
- The title attribute of <branch> should be concise (not greater than 10 characters), significantly shorter than the <branch> tag's content. The title can come from original parallel content if it meets conciseness requirements and summarizes the branch's core content; otherwise, you should generate it.
- If the original output can't be parallelized or is very short (not greater than 20 characters), directly include the original output without parallel tags.
- If the original output can be parallelized with many branches (>10), you may appropriately merge different parallel branch content and regenerate a suitable parallel title attribute, ensuring merged branches have similar lengths.
- <branchgroup> tags can appear multiple times, allowing parallel text to follow a format of: serial-parallel-serial-parallel-...
- If an <branchgroup> contains only one parallel branch, parallelization isn't needed - remove the corresponding <branchgroup> and <branch> tags.
- Ensure that parallel text with all parallel tags removed is coherent and must match the original output's expressed content and format.
- If generating parallel text, the language must match the original output text.

Here are a few input-output examples for your reference:

Input-Output Example 1:

[Original Output]

[INPUT\_START]

To connect and use a Lenovo wireless mouse and keyboard, follow these steps:

1. **Connect the Wireless Mouse**:
  - Ensure the mouse has sufficient battery power and turn it on.
  - Insert the receiver into the computer's USB port.
  - Press and hold the mouse scroll wheel while turning it to ON, hold for 3 seconds until the indicator light flashes, then bring it close to the receiver to complete pairing [1]([ref](#)).
2. **Connect the Wireless Keyboard**:
  - Ensure the keyboard has sufficient battery power and turn it on.
  - Insert the receiver into the computer's USB port.
  - Simultaneously press the "F2," "F3," and "F4" keys on the keyboard, turn it to ON, release the keys, and press "3." The rapid flashing of the indicator light indicates successful pairing [1]([ref](#)).

By following these steps, you can successfully connect and use Lenovo's wireless mouse and keyboard

[1,4]([ref](#)).

[INPUT\_END]

[Parallel Text]

[PARALLEL]

true

[PARALLEL]

[OUTPUT\_START]

To connect and use a Lenovo wireless mouse and keyboard, follow these steps:

- ```
<branchgroup><branch title="Wireless Mouse">1. Connect the Wireless Mouse:
  - Ensure the mouse has sufficient battery power and turn it on.
  - Insert the receiver into the computer's USB port.
  - Press and hold the mouse scroll wheel while turning it to ON, hold for 3 seconds until the
  indicator light flashes, then bring it close to the receiver to complete pairing
  [1](ref).</branch>

<branch title="Wireless Keyboard">2. Connect the Wireless Keyboard:
  - Ensure the keyboard has sufficient battery power and turn it on.
  - Insert the receiver into the computer's USB port.
  - Simultaneously press the "F2," "F3," and "F4" keys on the keyboard, turn it to ON, release the
  keys, and press "3." The rapid flashing of the indicator light indicates successful pairing
  [1](ref).</branch></branchgroup>
```

By following these steps, you can successfully connect and use Lenovo's wireless mouse and keyboard

[1,4]([ref](#)).

[OUTPUT\_END]

Input-Output Example 2:

[Original Output]

[INPUT\_START]

NetEase Youdao Notes supports importing various file formats. Users can import Word files, Evernote files, Youdao Notes files, and files from other Youdao Notes accounts [6,7]([ref](#)).

[INPUT\_END]

[Parallel Text]

[PARALLEL]

false

[PARALLEL]

[OUTPUT\_START]

[OUTPUT\_END]

972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025

```

Input-Output Example 3:
[Original Output]
[INPUT_START]
**Yes**, the changes in the proportion of ZTE's carrier network business revenue in its Q1 2024
financial report **are noteworthy**. Although direct data on the proportion change is not mentioned,
the company's strategic adjustments in the carrier network business and their impact on the overall
business are points worth noting. Below is a detailed introduction to the relevant information:

#### Overview of ZTE's Q1 2024 Financial Performance
- The company achieved operating revenue of 30.58 billion yuan, a year-on-year increase of 4.9%
[4,5] (@ref).
- Net profit attributable to the parent company was 2.74 billion yuan, a year-on-year increase of
3.7% [4] (@ref).

#### Performance of Business Segments
- Government and enterprise as well as consumer businesses have returned to a rapid growth trajectory
[4,5] (@ref).
- The company is accelerating its shift from full connectivity to "connectivity + computing power,"
fully expanding market space [4] (@ref). This indicates that ZTE has made strategic adjustments in its
carrier network business to adapt to market and technological trends.

#### Impact on ZTE's Overall Financial Condition
- These changes show that while maintaining steady growth, ZTE is seeking new growth opportunities
and market potential through strategic adjustments.
- The rapid growth of government and enterprise as well as consumer businesses, along with the
strategic adjustments in the carrier network business, positively impact ZTE's overall financial
condition, contributing to the company's long-term development and competitiveness.

In summary, although the specific data on the proportion change in ZTE's carrier network business
revenue in its Q1 2024 financial report is not directly mentioned, the company's strategic
adjustments and the performance of its business segments demonstrate its significant impact on the
overall financial condition and long-term development strategy, making it noteworthy.
[INPUT_END]

[Parallel Text]
[PARALLEL]
true
[PARALLEL]
[OUTPUT_START]
**Yes**, the changes in the proportion of ZTE's carrier network business revenue in its Q1 2024
financial report **are noteworthy**. Although direct data on the proportion change is not mentioned,
the company's strategic adjustments in the carrier network business and their impact on the overall
business are points worth noting. Below is a detailed introduction to the relevant information:

<branchgroup><branch title="Financial Performance">#### Overview of ZTE's Q1 2024 Financial
Performance
- The company achieved operating revenue of 30.58 billion yuan, a year-on-year increase of 4.9%
[4,5] (@ref).
- Net profit attributable to the parent company was 2.74 billion yuan, a year-on-year increase of
3.7% [4] (@ref).</branch>

<branch title="Business Segments">#### Performance of Business Segments
- Government and enterprise as well as consumer businesses have returned to a rapid growth trajectory
[4,5] (@ref).
- The company is accelerating its shift from full connectivity to "connectivity + computing power,"
fully expanding market space [4] (@ref). This indicates that ZTE has made strategic adjustments in its
carrier network business to adapt to market and technological trends.
</branch></branchgroup>

#### Impact on ZTE's Overall Financial Condition
- These changes show that while maintaining steady growth, ZTE is seeking new growth opportunities
and market potential through strategic adjustments.
- The rapid growth of government and enterprise as well as consumer businesses, along with the
strategic adjustments in the carrier network business, positively impact ZTE's overall financial
condition, contributing to the company's long-term development and competitiveness.
In summary, although the specific data on the proportion change in ZTE's carrier network business
revenue in its Q1 2024 financial report is not directly mentioned, the company's strategic
adjustments and the performance of its business segments demonstrate its significant impact on the
overall financial condition and long-term development strategy, making it noteworthy.
[OUTPUT_END]

Input-Output Example 4:
[Original Output]
[INPUT_START]
Alright, we first need to calculate the result of  $54 \times 12 + 23 \times 23$ , then perform the @ operation on each
number in the expression and the final result, and finally sum them up to get the final result.
We start by solving the expression  $54 \times 12 + 23 \times 23$ , which can be calculated step by step as follows:
1. Calculate 54 multiplied by 12.
2. Calculate 23 multiplied by 23.
3. Add the two results obtained above.

First, let's calculate each multiplication result:
### Calculate 54 multiplied by 12
1. Calculate 54 multiplied by 10.
 $54 \times 10 = 540$ 
2. Calculate 54 multiplied by 2.
 $54 \times 2 = 108$ 
3. Add the results of 54 multiplied by 10 and 54 multiplied by 2.
 $540 + 108 = 648$ 

### Calculate 23 multiplied by 23

```

1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079

```

1. Calculate 23 multiplied by 20.
23 * 20 = 460
2. Calculate 23 multiplied by 3.
23 * 3 = 69
3. Add the results of 23 multiplied by 20 and 23 multiplied by 3.
460 + 69 = 529

### Add the results of the two multiplications
1. 648 + 529 = 1177

Next, we need to perform the @ operation on each number in the expression (54, 12, 23, 23) and the
final result (1177):
### 54@1177
- Perform the @ operation on 54 and 1177:
\[ 54@1177 = 54^2 \times 1177 + 1 = 2916 \times 1177 + 1 = 3430932 + 1 = 3430933 \]
### 12@1177
- Perform the @ operation on 12 and 1177:
\[ 12@1177 = 12^2 \times 1177 + 1 = 144 \times 1177 + 1 = 169488 + 1 = 169489 \]
### 23@1177
- Perform the @ operation on 23 and 1177 (twice, since 23 appears twice):
\[ 23@1177 = 23^2 \times 1177 + 1 = 529 \times 1177 + 1 = 622033 + 1 = 622034 \]
\[ 23@1177 = 23^2 \times 1177 + 1 = 529 \times 1177 + 1 = 622033 + 1 = 622034 \]

Sum all the results of the @ operations:
\[ 3430933 + 169489 + 622034 + 622034 = 4840490 \]

The final result is 4840490.
[INPUT_END]

[Parallel Text]
[PARALLEL]
true
[PARALLEL]
[OUTPUT_START]
Alright, we first need to calculate the result of 54*12 + 23*23, then perform the @ operation on each
number in the expression and the final result, and finally sum them up to get the final result.
We start by solving the expression 54*12 + 23*23, which can be calculated step by step as follows:
1. Calculate 54 multiplied by 12.
2. Calculate 23 multiplied by 23.
3. Add the two results obtained above.

First, let's calculate each multiplication result:
<branchgroup><branch title="54*12">### Calculate 54 multiplied by 12
1. Calculate 54 multiplied by 10.
54 * 10 = 540
2. Calculate 54 multiplied by 2.
54 * 2 = 108
3. Add the results of 54 multiplied by 10 and 54 multiplied by 2.
540 + 108 = 648</branch>
<branch title="23*23">### Calculate 23 multiplied by 23
1. Calculate 23 multiplied by 20.
23 * 20 = 460
2. Calculate 23 multiplied by 3.
23 * 3 = 69
3. Add the results of 23 multiplied by 20 and 23 multiplied by 3.
460 + 69 = 529</branch></branchgroup>

### Add the results of the two multiplications
1. 648 + 529 = 1177

Next, we need to perform the @ operation on each number in the expression (54, 12, 23, 23) and the
final result (1177):
<branchgroup><branch title="54@1177">### 54@1177
- Perform the @ operation on 54 and 1177:
\[ 54@1177 = 54^2 \times 1177 + 1 = 2916 \times 1177 + 1 = 3430932 + 1 = 3430933 \]</branch>
<branch title="12@1177">### 12@1177
- Perform the @ operation on 12 and 1177:
\[ 12@1177 = 12^2 \times 1177 + 1 = 144 \times 1177 + 1 = 169488 + 1 = 169489 \]</branch>
<branch title="23@1177">### 23@1177
- Perform the @ operation on 23 and 1177 (twice, since 23 appears twice):
\[ 23@1177 = 23^2 \times 1177 + 1 = 529 \times 1177 + 1 = 622033 + 1 = 622034 \]
\[ 23@1177 = 23^2 \times 1177 + 1 = 529 \times 1177 + 1 = 622033 + 1 = 622034 \]
\]</branch></branchgroup>

Sum all the results of the @ operations:
\[ 3430933 + 169489 + 622034 + 622034 = 4840490 \]

The final result is 4840490.
[OUTPUT_END]

Now given the following original output, please output the corresponding parallel text according to
the above examples and requirements, and keep the original text while maintaining the output text in
the same language:
[[Original Output]]
[INPUT_START]
{answer}
[INPUT_END]

[[Parallel Text]]

```



## Independence Verification

Your task is to determine whether the subsequent parallel branches meet the condition of being independent and mutually non-dependent based on the output of parallel decoding--specifically, the question and the precondition text. This assessment is to judge the rationality of parallelization.

The background of parallel decoding is as follows: Certain content in the original output text of a language model may be suitable for conversion into a parallel format (where some parts are divided into multiple parallelizable blocks, and these blocks are logically independent and do not depend on each other). Parallel text generation allows the model to generate these contents simultaneously rather than sequentially, significantly improving decoding efficiency and response time. The general process of parallel decoding is as follows:

1. Serial content
2. Generate parallel content, which may consist of multiple parallel branches.
3. Wait for all parallel branches to complete before continuing with serial content.
4. Repeat steps 1-3 until the output is complete.

The core idea of parallel decoding is to split the original output text into different parallel branches where these branches can be completed independently without relying on each other's results. After all parallel branches are completed, the results are aggregated, and further parallel branch planning is performed until the problem is resolved. This division of parallel branches can be applied to tasks such as reading comprehension or mathematical problems.

Note the input and output formats are as follows:

```

**Input Format:**
[[Question]]
[TEXT_START]
XXXX (question)
[TEXT_END]
[[Precondition Text]]
[TEXT_START]
XXXX (content preceding the parallel branches)
[TEXT_END]
[[Parallel Text]]
[TEXT_START]
<Branch1 title="XX">
XXXXX (content of Branch1)
</Branch1>
<Branch2 title="XX">
XXXXX (content of Branch2)
</Branch2>
...
[TEXT_END]

**Output Format:**
[[Judgment Result]]
[PARALLEL]
X (parallel rationality; if the parallel branches are reasonably divided, independent, and mutually
non-dependent, X=true; otherwise, X=false)
[PARALLEL]
[REASON_START]
XXXX (reasoning for whether parallelization is reasonable)
[REASON_END]

```

After understanding the above background and input/output formats, you need to evaluate whether the branching content in the parallel text is reasonable based on the precondition text. The precondition text can be understood as the serial content in step 1 of the parallel decoding process, while the branching content consists of multiple parallel branches generated after the serial content. These parallel branches can access all information from the precondition text but cannot see information from other parallel branches. Note that in the input parallel content, each branch is wrapped in tags like `

For judging the rationality of parallelization, you must adhere to the following rules:

1. If any branch explicitly or implicitly relies on or uses intermediate results or conclusions from other branches, the parallel branching is unreasonable because it violates the principle of mutual independence.
2. If any branch depends on another branch as a precondition, the parallel division is also unreasonable.
3. If the `title` attribute of any parallel branches are identical, this would prevent parallel decoding from generating distinct parallel branches based on the titles, making the parallel division unreasonable.

Below are some input/output examples for reference:

```

**Input/Output Example 1:**
**Input:**
[[Question]]
[TEXT_START]
A driver travels for $2$ hours at $60$ miles per hour, during which her car gets $30$ miles per
gallon of gasoline. She is paid $\$0.50$ per mile, and her only expense is gasoline at $\$2.00$ per
gallon. What is her net rate of pay, in dollars per hour, after this expense?
[TEXT_END]
[[Precondition Text]]
[TEXT_START]
The driver travels for 2 hours at 60 miles per hour, so the distance traveled is:
60 miles/hour × 2 hours = 120 miles.
[TEXT_END]
[[Parallel Text]]
[TEXT_START]
<Branch1 title="Gross Pay">Her gross pay is calculated at $0.50 per mile:

```

```

1134
1135 120 miles × $0.50/mile = $60.00.
1136 </Branch1>
1137 <Branch2 title="Gasoline Usage">The car gets 30 miles per gallon, and she drove 120 miles, so the
1138 gasoline used is:
1139 120 miles / 30 miles/gallon = 4 gallons.
1140 </Branch2>
1141 [TEXT_END]
1142 **Output:**
1143 [[Judgment Result]]
1144 [PARALLEL]
1145 true
1146 [PARALLEL]
1147 [REASON_START]
1148 In both branches, the calculations for Gross Pay and Gasoline Usage do not explicitly or implicitly
1149 depend on each other or rely on the other's results, so the parallel division is reasonable.
1150 [REASON_END]
1151
1152 **Input/Output Example 2:**
1153 **Input:**
1154 [[Question]]
1155 [TEXT_START]
1156 Let  $f$  be the function defined by  $f(x)=ax^2-\sqrt{x}$  for some positive  $a$ . If
1157  $f(\sqrt{x})=-\sqrt{x}$  then  $a=$ 
1158 [TEXT_END]
1159 [[Precondition Text]]
1160 [TEXT_START]
1161 The function is defined as  $(f(x) = ax^2 - \sqrt{x})$ , where  $(a > 0)$ . The given condition is
1162  $(f(\sqrt{x}) = -\sqrt{x})$ .
1163 First, calculate  $(f(\sqrt{x}))$ :
1164  $(f(\sqrt{x}) = a(\sqrt{x})^2 - \sqrt{x} = a \cdot 2 - \sqrt{x} = 2a - \sqrt{x})$ .
1165 Next, calculate  $(f(f(\sqrt{x}))) = f(2a - \sqrt{x})$ :
1166  $(f(2a - \sqrt{x}) = a(2a - \sqrt{x})^2 - \sqrt{2a - \sqrt{x}})$ .
1167 Expand  $((2a - \sqrt{x})^2)$ :
1168  $((2a - \sqrt{x})^2 = (2a)^2 - 2 \cdot (2a) \cdot (\sqrt{x}) + (\sqrt{x})^2 = 4a^2 - 4a\sqrt{x} + 2)$ .
1169 Substitute back into the original equation:
1170  $(f(2a - \sqrt{x}) = a(4a^2 - 4a\sqrt{x} + 2) - \sqrt{2a - \sqrt{x}} = 4a^3 - 4a^2\sqrt{x} + 2a - \sqrt{2a - \sqrt{x}})$ .
1171 Set this equal to  $(-\sqrt{x})$ :
1172  $(4a^3 - 4a^2\sqrt{x} + 2a - \sqrt{2a - \sqrt{x}} = -\sqrt{x})$ .
1173 Add  $(\sqrt{2a - \sqrt{x}})$  to both sides:
1174  $(4a^3 - 4a^2\sqrt{x} + 2a = 0)$ .
1175 Factor out  $(a)$  (since  $(a > 0)$ ,  $(a \neq 0)$ ):
1176  $(a(4a^2 - 4a\sqrt{x} + 2) = 0)$ .
1177 Solve for  $(a)$ :
1178  $(4a^2 - 4a\sqrt{x} + 2 = 0)$ .
1179 Divide by 2 to simplify:
1180  $(2a^2 - 2a\sqrt{x} + 1 = 0)$ .
1181 Use the quadratic formula to solve:
1182  $(a = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a})$ , where  $(A = 2)$ ,  $(B = -2\sqrt{x})$ ,  $(C = 1)$ .
1183 The discriminant is:
1184  $(D = (-2\sqrt{x})^2 - 4 \cdot 2 \cdot 1 = 8 - 8 = 0)$ .
1185 Since  $(D = 0)$ , there is only one real solution:
1186  $(a = \frac{2\sqrt{x}}{2} = \sqrt{x})$ .
1187 Verify the solution:
1188 Let  $(a = \sqrt{x})$ .
1189 [TEXT_END]
1190 [[Parallel Text]]
1191 [TEXT_START]
1192 <Branch1 title="Calculate  $f(\sqrt{x})$ ">Calculate  $(f(\sqrt{x}))$ :
1193  $(f(\sqrt{x}) = \frac{\sqrt{x}}{2} \cdot (\sqrt{x})^2 - \sqrt{x} = \frac{\sqrt{x}}{2} \cdot 2 - \sqrt{x} = \sqrt{x} - \sqrt{x} = 0)$ .
1194 </Branch1>
1195 <Branch2 title="Calculate  $f(0)$ ">Then calculate  $(f(f(\sqrt{x}))) = f(0)$ :
1196  $(f(0) = \frac{\sqrt{x}}{2} \cdot 0^2 - \sqrt{x} = -\sqrt{x})$ .
1197 </Branch2>
1198 [TEXT_END]
1199 **Output:**
1200 [[Judgment Result]]
1201 [PARALLEL]
1202 false
1203 [PARALLEL]
1204 [REASON_START]
1205 In Branch2, "Then calculate  $(f(f(\sqrt{x}))) = f(0)$ " clearly relies on the conclusion  $(f(\sqrt{x}) = 0)$  from Branch1. Thus, Branch2 depends on Branch1's result, making the parallel
1206 division unreasonable.
1207 [REASON_END]
1208
1209 Now, given the following question, precondition text, and parallel content, please provide a judgment
1210 on the rationality of parallelization based on the above examples and criteria:
1211 [[Question]]
1212 [TEXT_START]
1213 {question}
1214 [TEXT_END]
1215 [[Precondition Text]]
1216 [TEXT_START]
1217 {answer_prefix}
1218 [TEXT_END]
1219 [[Parallel Text]]
1220 [TEXT_START]
1221 {branch_infos}
1222 [TEXT_END]
1223
1224 [[Judgment Result]]

```

1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
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

## Integrity Verification

Your task is to compare whether the model answer's format matches the standard answer's format and integrity.

Note that the input and output formats are as follows:

Input format:

```
[[Standard Answer]]
[TEXT_START]
XXXX (Standard answer content)
[TEXT_END]
[[Model Answer]]
[TEXT_START]
XXXX (Model answer content)
[TEXT_END]
```

Output format:

```
[[Judgment Result]]
[SAME_FOREMAT]
X (Whether the formats match. X=true if the model answer's format matches the standard answer's;
otherwise, X=false)
[SAME_FOREMAT]
[REASON_START]
XXXX (Reason for whether the formats match)
[REASON_END]
```

After understanding the background and input-output format above, you need to determine whether the model answer's format and integrity matches the standard answer's. These formats may include markdown, JSON, HTML, etc. For judging format consistency, you must follow the rules below:

1. If the standard answer content has specific formatting, such as specific text or non-whitespace symbols before paragraphs, or the text is divided into multiple sections with certain non-whitespace symbols, and the model answer lacks corresponding formatting, then the formats are judged as inconsistent.
2. If the standard answer contains JSON content, and the model answer's JSON has even one differing key or value, the formats are judged as inconsistent. Additionally, the model answer's JSON must comply with JSON formatting standards; otherwise, it is also considered inconsistent.
3. If the standard answer contains markdown content, and the model answer differs in heading levels, numbering quantity, numbering format, or text formatting (bold, underline, etc.), the formats are judged as inconsistent.
4. If the standard answer contains HTML or XML content, you must check whether the model answer has the corresponding number of HTML/XML tags and whether the nesting hierarchy matches. If any requirement is unmet, the formats are judged as inconsistent.
5. If the model answer's key points (i.e., the semantic content describing the subject, explanations, or conclusions) are fewer or more than the standard answer's, the formats are also judged as inconsistent.
6. Note that blank lines, line breaks, and whitespace characters in the model answer do not affect the judgment of format consistency.

Below are some input-output examples for reference:

Input-Output Example 1:

```
[[Standard Answer]]
[TEXT_START]
To use the Game Boost feature on Xiaomi phones, follow these steps:
1. **Open Security Center**: Find and tap the "Security Center" app on the home screen [7](@ref).
2. **Enter Game Boost**: In the Security Center interface, tap the "Game Boost" option [1,3](@ref).
3. **Add a Game**: After entering the Game Boost interface, tap the "+" icon in the top-right corner to select and add the game you want to boost [5](@ref).
4. **Optimize Settings**: Tap the "Gear" icon to enter the settings page, where you can enable features like memory cleanup, network acceleration, and anti-misoperation [4,6](@ref).
```

These steps can effectively improve game smoothness and reduce interruptions from calls and notifications during gameplay [3,7](@ref).

```
[TEXT_END]
[[Model Answer]]
[TEXT_START]
To use the Game Boost feature on Xiaomi phones, follow these steps:
```

1. Open Security Center: Find and tap the "Security Center" app on the home screen [7](@ref).
2. Enter Game Boost: In the Security Center interface, tap the "Game Boost" option [1,3](@ref).
3. Add a Game: After entering the Game Boost interface, tap the "+" icon in the top-right corner to select and add the game you want to boost [5](@ref).
4. Optimize Settings: Tap the "Gear" icon to enter the settings page, where you can enable features like memory cleanup, network acceleration, and anti-misoperation [4,6](@ref).

These steps can effectively improve game smoothness and reduce interruptions from calls and notifications during gameplay [3,7](@ref).

```
[TEXT_END]
[[Judgment Result]]
[SAME_FOREMAT]
false
[SAME_FOREMAT]
[REASON_START]
The model answer lacks bold formatting for the step descriptions, making it inconsistent with the standard answer's format.
[REASON_END]
```

Input-Output Example 2:

```
[[Standard Answer]]
[TEXT_START]
```

1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295

When a real-name-verified primary account logs into the large model knowledge engine product for the first time, the following free trial quotas are available:

- Fine-tuned Large Model Standard Edition: Receive 500,000 free tokens, valid for 2 months.
- Knowledge Base Capacity: Receive 3,000,000 free characters, valid for 6 months.
- Atomic Capability - Multi-turn Rewrite: Receive 500,000 free tokens for multi-turn rewriting, valid for 2 months.
- Atomic Capability - Embedding: Receive 500,000 free tokens for embedding, valid for 2 months.

Note: Free resource packs are deducted first, with a shared quota of 500,000 tokens.

[TEXT\_END]  
[[Model Answer]]  
[TEXT\_START]

When a real-name-verified primary account logs into the large model knowledge engine product for the first time, the following free trial quotas are available:

1. Fine-tuned Large Model Standard Edition:  
Receive 500,000 free tokens, valid for 2 months.
2. Knowledge Base Capacity:  
Receive 3,000,000 free characters, valid for 6 months.
3. Atomic Capability - Multi-turn Rewrite:  
Receive 500,000 free tokens for multi-turn rewriting, valid for 2 months.
4. Atomic Capability - Embedding:  
Receive 500,000 free tokens for embedding, valid for 2 months.

Note: Free resource packs are deducted first, with a shared quota of 500,000 tokens.

[TEXT\_END]  
[[Judgment Result]]  
[SAME\_FOREMAT]  
false  
[SAME\_FOREMAT]  
[REASON\_START]

The model answer replaces the standard answer's "-" symbols with numbered lists, making the format inconsistent.

[REASON\_END]

Input-Output Example 3:

[[Standard Answer]]  
[TEXT\_START]  
Thought:

Convert "new york city" to title case, resulting in 'New York City'.

Final Answer:

```
{"No. 1": {"City": "New York City"}}
[TEXT_END]
[[Model Answer]]
[TEXT_START]
```

Convert "new york city" to title case, resulting in 'New York City'.

Final Answer:

```
{"No. 1": {"City": "New York City"}}
[TEXT_END]
[[Judgment Result]]
[SAME_FOREMAT]
false
[SAME_FOREMAT]
[REASON_START]
```

The model answer omits the "Thought:" label in the first paragraph, making it inconsistent with the standard answer's format.

[REASON\_END]

Input-Output Example 4:

[[Standard Answer]]  
[TEXT\_START]

1. **Create Contract**: Sales representatives search for "Contracts" in App Launcher, click "New," enter contract details, and save changes [3](@ref).
2. **Link to Opportunity**: Ensure the contract appears in the opportunity's details for quick access by sales reps [3](@ref).
3. **Automation and Reminders**: Set workflow alerts to remind sales reps of contract renewals and other key actions [3](@ref).
4. **E-Signature**: Use tools like DocuSign, integrated with Salesforce, to simplify the signing process [4](@ref).

[TEXT\_END]  
[[Model Answer]]  
[TEXT\_START]

1. **Create Contract**: Sales representatives search for "Contracts" in App Launcher, click "New," enter contract details, and save changes [3](@ref).
2. **Link to Opportunity**: Ensure the contract appears in the opportunity's details for quick access by sales reps [3](@ref).
3. **Automation and Reminders**: Set workflow alerts to remind sales reps of contract renewals and other key actions [3](@ref).
4. **E-Signature**: Use tools like DocuSign, integrated with Salesforce, to simplify the signing process [4](@ref).

[TEXT\_END]  
[[Judgment Result]]  
[SAME\_FOREMAT]  
true  
[SAME\_FOREMAT]  
[REASON\_START]

1296

1297 The model answer matches the standard answer in both content and format.  
[REASON\_END]

1298

1299 Input-Output Example 5:  
[[Standard Answer]]  
[TEXT\_START]

1300 Final Answer:  
```json  
{"Date": "1965-11-19"}  
...  
[TEXT\_END]

1301

1302 [[Model Answer]]  
[TEXT\_START]

1303 Final Answer:  
```json  
...  
[TEXT\_END]

1304 [[Judgment Result]]  
[SAME\_FOREMAT]  
false  
[SAME\_FOREMAT]

1305

1306 [REASON\_START]  
The model answer omits the JSON content (the part after ```json) present in the standard answer,  
making the format inconsistent.  
[REASON\_END]

1307

1308

1309

1310

1311

1312 Now, given the following standard answer and model answer, please provide the format consistency  
judgment result based on the examples and format consistency rules above:

1313 [[Standard Answer]]  
[TEXT\_START]  
{raw\_answer}  
[TEXT\_END]

1314

1315 [[Model Answer]]  
[TEXT\_START]  
{model\_answer}  
[TEXT\_END]

1316

1317

1318 [[Judgment Result]]  
\end{verbatim}  
\end{tcolorbox}

1319

1320 \begin{tcolorbox}[fontupper=\ttfamily\tiny,title={Answer  
Verification},width=\textwidth,colback=red!5,colframe=red!75!black]

1321 \begin{verbatim}[breaklines=true,breaksymbolleft={},breaksymbolright={}]

1322 Your task is to compare whether the model answer is consistent with the standard answer in terms of  
content and conclusion, i.e., answer consistency.

1323

1324 Note that the input and output formats are as follows:

1325 Input format:  
[[Standard Answer]]  
[TEXT\_START]  
XXXX (content of the standard answer)  
[TEXT\_END]

1326

1327 [[Model Answer]]  
[TEXT\_START]  
XXXX (content of the model answer)  
[TEXT\_END]

1328

1329

1330 Output format:  
[[Judgment Result]]  
[SAME\_ANSWER]  
X (whether the answers are consistent; X=true if the model answer and standard answer are consistent  
in content and conclusion, otherwise X=false)  
[SAME\_ANSWER]  
[REASON\_START]  
XXXX (reason for consistency or inconsistency)  
[REASON\_END]

1331

1332

1333

1334

1335

1336 After understanding the background and input/output formats above, you need to determine whether the  
model answer and standard answer are consistent in content and conclusion, following these rules:

1337 1. If the model answer and standard answer differ in the final answer/conclusion, then the answers  
are inconsistent.

1338 2. If the key points in the model answer (key points refer to the semantic content described in the  
text, i.e., the described objects, explanations, conclusions, etc.) are fewer or more than those in  
the standard answer, then the answers are also judged as inconsistent.

1339 3. The wording of the model answer and standard answer may differ, i.e., there can be some variation  
in content, but the key points mentioned in Rule 2 must be reflected (comparison is based on  
semantics, not verbatim). Otherwise, the answers can also be judged as inconsistent.

1340 4. Note that line breaks and whitespace characters in the model answer do not affect the judgment of  
answer consistency.

1341 5. If the model answer and standard answer are in different languages, then the answers are  
inconsistent.

1342

1343

1344 Below are some input/output examples for reference:

1345

1346 Input/Output Example 1:  
[[Standard Answer]]  
[TEXT\_START]

1347 To use the Game Turbo feature on Xiaomi phones, follow these steps:

1348 1. **Open Security Center**: Find and tap the "Security Center" app on the phone's interface  
[7](@ref).

1349

1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403

2. **\*\*Enter Game Turbo\*\***: In the Security Center interface, tap the "Game Turbo" option [1,3](@ref).
3. **\*\*Add Games\*\***: After entering the Game Turbo interface, tap the "+" icon in the upper-right corner to select and add the games you want to accelerate [5](@ref).
4. **\*\*Optimize Settings\*\***: Tap the "gear" icon to enter the settings page, where you can enable features like memory cleanup, network acceleration, and anti-misoperation [4,6](@ref).

These steps can effectively improve game smoothness and reduce interruptions from calls and notifications during gameplay [3,7](@ref).

[TEXT\_END]  
[[Model Answer]]  
[TEXT\_START]

To use the Game Turbo feature on Xiaomi phones, follow these steps:

1. Open Security Center: Find and tap the "Security Center" app on the phone's interface [7](@ref).
2. Enter Game Turbo: In the Security Center interface, tap the "Game Turbo" option [1,3](@ref).
3. Add Games: After entering the Game Turbo interface, tap the "+" icon in the upper-right corner to select and add the games you want to accelerate [5](@ref).

These steps can effectively improve game smoothness and reduce interruptions from calls and notifications during gameplay [3,7](@ref).

[TEXT\_END]  
[[Judgment Result]]  
[SAME\_ANSWER]

false  
[SAME\_ANSWER]  
[REASON\_START]  
The model answer lacks Step 4, so the answers are inconsistent.  
[REASON\_END]

Input/Output Example 2:

[[Standard Answer]]  
[TEXT\_START]

Thought:  
Convert "new york city" to capitalize the first letter of each word and lowercase the rest, i.e., 'New York City'.

Final Answer:  
{"Entry1": {"City": "New York City"}}  
[TEXT\_END]

[[Model Answer]]  
[TEXT\_START]  
Convert "new york city" to capitalize the first letter of each word and lowercase the rest, i.e., 'New york City'.

Final Answer:  
{"Entry1": {"City": "New york City"}}  
[TEXT\_END]

[[Judgment Result]]  
[SAME\_ANSWER]  
false  
[SAME\_ANSWER]  
[REASON\_START]  
The model answer's conversion result for "new york city" is 'New york City', which differs from the standard answer 'New York City', so the answers are inconsistent.  
[REASON\_END]

Input/Output Example 3:

[[Standard Answer]]  
[TEXT\_START]

Regarding the question about genital herpes, based on the known information, I cannot provide a relevant answer. The reasons are as follows:

1. The known information focuses on legally notifiable diseases in the fever-with-rash syndrome, including measles, rubella, hand-foot-and-mouth disease, chickenpox, shingles, etc., but does not mention genital herpes.
2. Genital herpes falls under the category of sexually transmitted diseases (mainly caused by HSV-2), which is different from the varicella-zoster virus (VZV) and enterovirus (EV) covered in the current materials.
3. The vaccination plans in the existing materials (e.g., recombinant zoster vaccine) only target the varicella-zoster virus and do not involve prevention for genital herpes.

It is recommended to consult a professional medical institution or a dermatologist/venereologist for targeted diagnosis and treatment advice. For international logistics-related infectious disease prevention measures, we can provide professional advice, but this question is beyond our expertise.

[TEXT\_END]  
[[Model Answer]]  
[TEXT\_START]

Regarding the question about genital herpes, based on the known information, I cannot provide a relevant answer.

[TEXT\_END]  
[[Judgment Result]]  
[SAME\_ANSWER]

false  
[SAME\_ANSWER]  
[REASON\_START]  
The model answer lacks the specific reasons and subsequent recommendations provided in the standard answer.  
[REASON\_END]

Input/Output Example 4:

[[Standard Answer]]  
[TEXT\_START]



1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457

```
Based on the known information, the support status for the EasyMesh version of TL-XDR6010 is as follows:
1. **TL-XDR6010 EasyMesh Version 1.0**:  
Supports EasyMesh 2.0 out of the box.
2. **TL-XDR6010 EasyMesh Version 2.0**:  
Requires upgrading to firmware version 1.0.9 Build 20240205 Rel.35518 or later to support EasyMesh 2.0.
[TEXT_END]
[[Model Answer]]
[TEXT_START]
Apologies, the smart assistant cannot answer this question yet.
The known information cannot address this question.
[TEXT_END]
[[Judgment Result]]
[SAME_ANSWER]
false
[SAME_ANSWER]
[REASON_START]
The model answer and standard answer are inconsistent in their statements.
[REASON_END]

Input/Output Example 5:
[[Standard Answer]]
[TEXT_START]
Thought: The user expressed an intent to query the parking location of a vehicle, which aligns with the task information. However, the user did not provide any specific parameter information. According to the task requirements, the first mandatory parameter, "license plate number," must be inquired about first.
[TEXT_END]
Final Answer: {"Parameters": {"License Plate Number":"","Query Date":"","Query Time Range":""}, "Reply": "May I ask for your license plate number? The license plate number is the official registration number used to uniquely identify your vehicle, e.g., Yue B12345.", "Is Conversation Over": "No"}
[TEXT_END]
[[Model Answer]]
[TEXT_START]
Thought: The user expressed an intent to query the parking location of a vehicle, which aligns with the task information. However, the user did not provide any specific parameter information. Therefore, the first mandatory parameter, "license plate number," must be inquired about first.
Final Answer: {"Parameters":{"License Plate Number":"","Query Date":"","Query Time Range":"","Reply":"May I ask for your license plate number? The license plate number is the official registration number used to uniquely identify your vehicle, e.g., Yue B12345.", "Is Conversation Over": "No"}
[TEXT_END]
[[Judgment Result]]
[SAME_ANSWER]
true
[SAME_ANSWER]
[REASON_START]
The key points and Final Answer in the model answer are consistent with the standard answer, so the answer consistency is true.
[REASON_END]

Now, given the standard answer and model answer below, please provide the judgment result for answer consistency based on the examples and requirements above:
[[Standard Answer]]
[TEXT_START]
{raw_answer}
[TEXT_END]
[[Model Answer]]
[TEXT_START]
{model_answer}
[TEXT_END]

[[Judgment Result]]
```

### Answer Verification

```
Your task is to compare whether the model answer is consistent with the standard answer in terms of content and conclusion, i.e., answer consistency.

Note that the input and output formats are as follows:
Input format:
[[Standard Answer]]
[TEXT_START]
XXXX (content of the standard answer)
[TEXT_END]
[[Model Answer]]
[TEXT_START]
XXXX (content of the model answer)
[TEXT_END]

Output format:
[[Judgment Result]]
[SAME_ANSWER]
X (whether the answers are consistent; X=true if the model answer and standard answer are consistent in content and conclusion, otherwise X=false)
[SAME_ANSWER]
[REASON_START]
XXXX (reason for consistency or inconsistency)
[REASON_END]
```

1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511

After understanding the background and input/output formats above, you need to determine whether the model answer and standard answer are consistent in content and conclusion, following these rules:

1. If the model answer and standard answer differ in the final answer/conclusion, then the answers are inconsistent.
2. If the key points in the model answer (key points refer to the semantic content described in the text, i.e., the described objects, explanations, conclusions, etc.) are fewer or more than those in the standard answer, then the answers are also judged as inconsistent.
3. The wording of the model answer and standard answer may differ, i.e., there can be some variation in content, but the key points mentioned in Rule 2 must be reflected (comparison is based on semantics, not verbatim). Otherwise, the answers can also be judged as inconsistent.
4. Note that line breaks and whitespace characters in the model answer do not affect the judgment of answer consistency.
5. If the model answer and standard answer are in different languages, then the answers are inconsistent.

Below are some input/output examples for reference:

Input/Output Example 1:

[[Standard Answer]]

[TEXT\_START]

To use the Game Turbo feature on Xiaomi phones, follow these steps:

1. **Open Security Center**: Find and tap the "Security Center" app on the phone's interface [7](@ref).
2. **Enter Game Turbo**: In the Security Center interface, tap the "Game Turbo" option [1,3](@ref).
3. **Add Games**: After entering the Game Turbo interface, tap the "+" icon in the upper-right corner to select and add the games you want to accelerate [5](@ref).
4. **Optimize Settings**: Tap the "gear" icon to enter the settings page, where you can enable features like memory cleanup, network acceleration, and anti-misoperation [4,6](@ref).

These steps can effectively improve game smoothness and reduce interruptions from calls and notifications during gameplay [3,7](@ref).

[TEXT\_END]

[[Model Answer]]

[TEXT\_START]

To use the Game Turbo feature on Xiaomi phones, follow these steps:

1. Open Security Center: Find and tap the "Security Center" app on the phone's interface [7](@ref).
2. Enter Game Turbo: In the Security Center interface, tap the "Game Turbo" option [1,3](@ref).
3. Add Games: After entering the Game Turbo interface, tap the "+" icon in the upper-right corner to select and add the games you want to accelerate [5](@ref).

These steps can effectively improve game smoothness and reduce interruptions from calls and notifications during gameplay [3,7](@ref).

[TEXT\_END]

[[Judgment Result]]

[SAME\_ANSWER]

false

[SAME\_ANSWER]

[REASON\_START]

The model answer lacks Step 4, so the answers are inconsistent.

[REASON\_END]

Input/Output Example 2:

[[Standard Answer]]

[TEXT\_START]

Thought:

Convert "new york city" to capitalize the first letter of each word and lowercase the rest, i.e., 'New York City'.

Final Answer:

{"Entry1": {"City": "New York City"}}

[TEXT\_END]

[[Model Answer]]

[TEXT\_START]

Convert "new york city" to capitalize the first letter of each word and lowercase the rest, i.e., 'New york City'.

Final Answer:

{"Entry1": {"City": "New york City"}}

[TEXT\_END]

[[Judgment Result]]

[SAME\_ANSWER]

false

[SAME\_ANSWER]

[REASON\_START]

The model answer's conversion result for "new york city" is 'New york City', which differs from the standard answer 'New York City', so the answers are inconsistent.

[REASON\_END]

Input/Output Example 3:

[[Standard Answer]]

[TEXT\_START]

Regarding the question about genital herpes, based on the known information, I cannot provide a relevant answer. The reasons are as follows:

1. The known information focuses on legally notifiable diseases in the fever-with-rash syndrome, including measles, rubella, hand-foot-and-mouth disease, chickenpox, shingles, etc., but does not mention genital herpes.

1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565

2. Genital herpes falls under the category of sexually transmitted diseases (mainly caused by HSV-2), which is different from the varicella-zoster virus (VZV) and enterovirus (EV) covered in the current materials.

3. The vaccination plans in the existing materials (e.g., recombinant zoster vaccine) only target the varicella-zoster virus and do not involve prevention for genital herpes.

It is recommended to consult a professional medical institution or a dermatologist/venerologist for targeted diagnosis and treatment advice. For international logistics-related infectious disease prevention measures, we can provide professional advice, but this question is beyond our expertise.

[[TEXT\_END]]  
[[Model Answer]]

[[TEXT\_START]]

Regarding the question about genital herpes, based on the known information, I cannot provide a relevant answer.

[[TEXT\_END]]

[[Judgment Result]]

[[SAME\_ANSWER]]

false

[[SAME\_ANSWER]]

[[REASON\_START]]

The model answer lacks the specific reasons and subsequent recommendations provided in the standard answer.

[[REASON\_END]]

Input/Output Example 4:

[[Standard Answer]]

[[TEXT\_START]]

Based on the known information, the support status for the EasyMesh version of TL-XDR6010 is as follows:

1. \*\*TL-XDR6010 EasyMesh Version 1.0\*\*:

Supports EasyMesh 2.0 out of the box.

2. \*\*TL-XDR6010 EasyMesh Version 2.0\*\*:

Requires upgrading to firmware version 1.0.9 Build 20240205

Rel.35518 or later to support EasyMesh 2.0.

[[TEXT\_END]]

[[Model Answer]]

[[TEXT\_START]]

Apologies, the smart assistant cannot answer this question yet.

The known information cannot address this question.

[[TEXT\_END]]

[[Judgment Result]]

[[SAME\_ANSWER]]

false

[[SAME\_ANSWER]]

[[REASON\_START]]

The model answer and standard answer are inconsistent in their statements.

[[REASON\_END]]

Input/Output Example 5:

[[Standard Answer]]

[[TEXT\_START]]

Thought: The user expressed an intent to query the parking location of a vehicle, which aligns with the task information. However, the user did not provide any specific parameter information. According to the task requirements, the first mandatory parameter, "license plate number," must be inquired about first.

Final Answer: {"Parameters": {"License Plate Number": "", "Query Date": "", "Query Time Range": ""}, "Reply": "May I ask for your license plate number? The license plate number is the official registration number used to uniquely identify your vehicle, e.g., Yue B12345.", "Is Conversation Over": "No"}

[[TEXT\_END]]

[[Model Answer]]

[[TEXT\_START]]

Thought: The user expressed an intent to query the parking location of a vehicle, which aligns with the task information. However, the user did not provide any specific parameter information.

Therefore, the first mandatory parameter, "license plate number," must be inquired about first.

Final Answer: {"Parameters": {"License Plate Number": "", "Query Date": "", "Query Time Range": ""}, "Reply": "May I ask for your license plate number? The license plate number is the official registration number used to uniquely identify your vehicle, e.g., Yue B12345.", "Is Conversation Over": "No"}

[[TEXT\_END]]

[[Judgment Result]]

[[SAME\_ANSWER]]

true

[[SAME\_ANSWER]]

[[REASON\_START]]

The key points and Final Answer in the model answer are consistent with the standard answer, so the answer consistency is true.

[[REASON\_END]]

Now, given the standard answer and model answer below, please provide the judgment result for answer consistency based on the examples and requirements above:

[[Standard Answer]]

[[TEXT\_START]]

{raw\_answer}

[[TEXT\_END]]

[[Model Answer]]

[[TEXT\_START]]

{model\_answer}

[[TEXT\_END]]

[[Judgment Result]]