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

Anonymous authors
Paper under double-blind review

ABSTRACT

The growing scale and complexity of large language models (LLMs) lead to high inference latency, primarily due to their autoregressive decoding characterized by the next-token prediction paradigm. By re-examining the outputs of autoregressive models, we find that some segments exhibit parallelizable structures, which we term intrinsic parallelism. Decoding these segments simultaneously (*i.e.* parallel decoding) can substantially accelerate LLM inference speed. In this paper, we propose an **A**daptive **S**erial-**P**arallel **D**ecoding (**ASPD**), which addresses two core challenges: automated construction of parallelizable data and efficient parallel decoding mechanism. 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 and Retrieval-Augmented Generation demonstrate that **ASPD** achieves unprecedented performance in both effectiveness and efficiency. Notably, on Vicuna Bench, our method achieves up to 3.10x speedup while maintaining response quality within 1% difference compared to autoregressive models, realizing significant acceleration without compromising generation quality.

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. 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.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

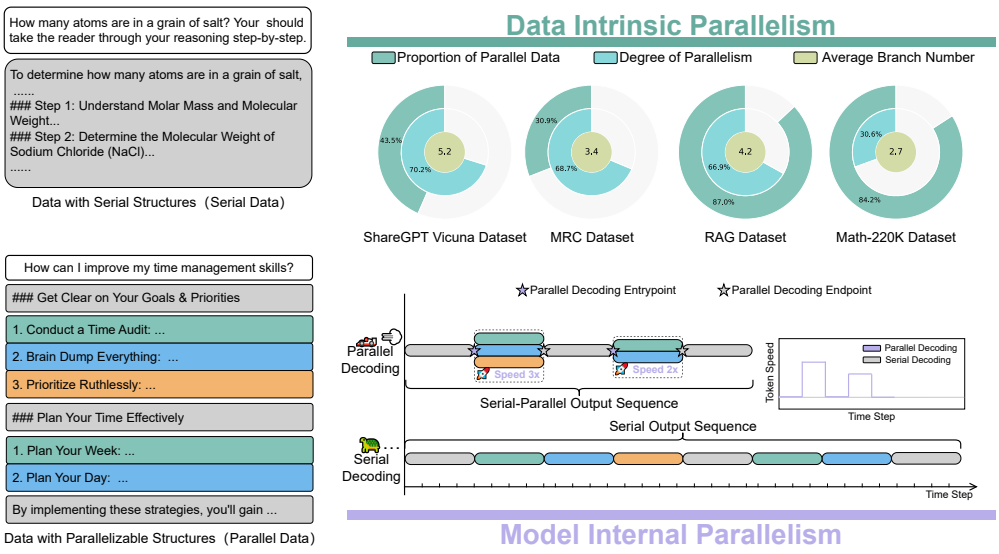


Figure 1: Overview of data intrinsic and model internal parallelism. The definition of Proportion of Parallel Data, Degree of Parallelism and Average Branch Number can be found in Section 4.1.

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 **A**daptive **S**erial-**P**arallel **D**ecoding, 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 general tasks (Vicuna Bench, MT Bench) and retrieval-augmented generation (Neural-Bridge-RAG) benchmarks - we demonstrate significant improvements in both effectiveness and efficiency compared to existing approaches, achieving an optimal balance between these two critical metrics.

2 Related Works

Existing research on parallelizing large language model inference can be categorized into two directions: enhancing response quality or accelerating generation speed. Quality-oriented methods

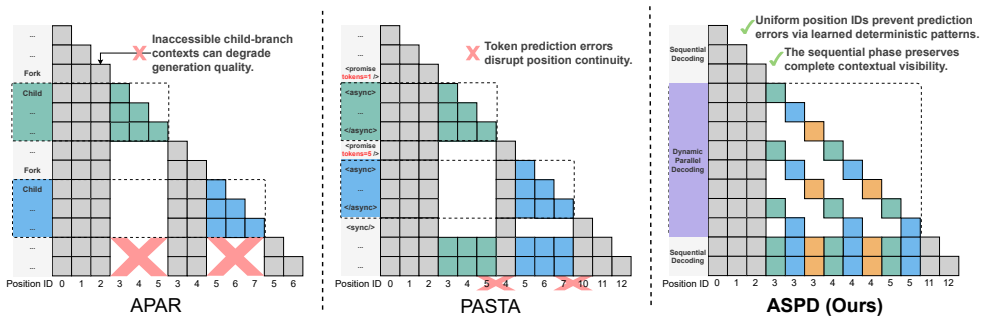


Figure 2: Related parallel decoding methods.

(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. **Orthogonal Acceleration Techniques** Speculative decoding 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. 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 2, 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.

3 METHODOLOGY

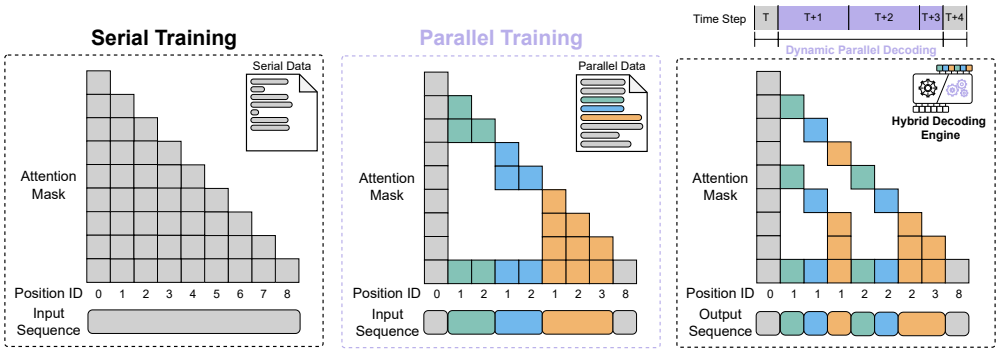


Figure 3: Serial-parallel training details (left-center). Hybrid Decoding Engine (right).

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

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

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 $\{A_{iv_1}, A_{iv_2}, \ldots, A_{iv_M}\}$ 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.

3.2 Internal Parallelization Module

Model Architecture for Native Parallelization

After constructing the parallelized corpus described in Section 3.1, we adapt the model architecture to support efficient transitions between parallel and sequential decoding. As illustrated in Figure 3, this requires controlling both inter-branch visibility and positional encoding within the parallel region. To preserve the capabilities of the native autoregressive model while enabling parallel decoding, we introduce two key components: (1) an internal parallel attention mask that prevents information leakage across branches, and (2) shared positional encodings for tokens generated at the same decoding step across different branches.

We further quantify the approximation introduced by parallel decoding through a likelihood-based analysis. Let the output sequence Y consist of a prefix P, k branches $B = \{B_1, \ldots, B_k\}$, and a suffix S. A standard autoregressive model factorizes the conditional probability as $P_{\mathrm{AR}}(Y \mid X) = P(P \mid X) \left(\prod_{i=1}^{k} P(B_i \mid B_{<i}, P, X) \right) P(S \mid B, P, X)$. In ASPD, the branches are approximated as conditionally independent given the prefix and input: $P_{\mathrm{ASPD}}(Y \mid X) = P(P \mid X) \left(\prod_{i=1}^{k} P(B_i \mid P, X) \right) P(S \mid B, P, X)$. The suffix remains autoregressive and can attend to all completed branches; therefore, the approximation is confined to the parallel branch region. Assuming that ASPD preserves the corresponding branch marginals, the resulting divergence is the conditional total correlation among the branches: $\mathcal{L}_{\mathrm{approx}} = D_{\mathrm{KL}} \left(P_{\mathrm{AR}}(B \mid P, X) \,\middle\|\, \prod_{i=1}^{k} P_{\mathrm{AR}}(B_i \mid P, X) \right) = \mathrm{TC}(B_1, \ldots, B_k \mid P, X)$. Here, TC denotes the conditional total correlation, which quantifies inter-branch dependence given P and X, reducing to conditional mutual information when $k = 2$. ASPD minimizes this dependence through branch verification, thereby limiting approximation error while accelerating decoding.

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 Δi represents the relative position of token i within its stage. P_t represents the number of tokens being decoded simultaneously at time t.

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

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

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

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

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 $< b(i), b(j) >$ 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 3, 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

Table 1: Performance comparison on MT Bench and Vicuna Bench.

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

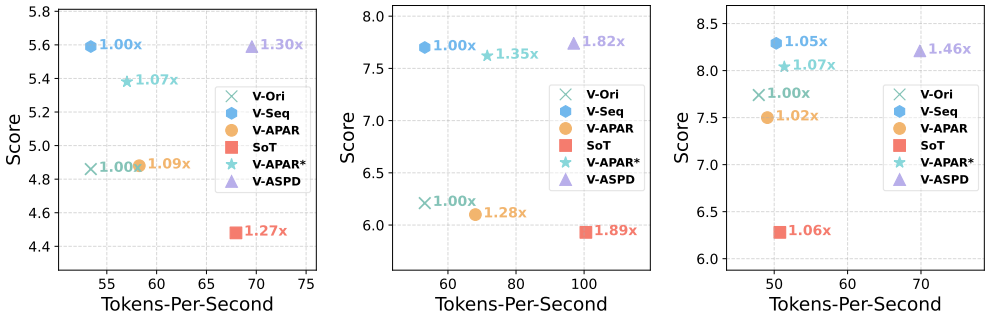
4.1 Experimental Setup

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). 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 **T**okens-**P**er-**S**econd (**TPS**) as the primary throughput metric. For parallel models, we further introduce four additional metrics to characterize parallelization performance: **P**arallel-**T**okens-**P**er-**S**econd (**P-TPS**, TPS in parallel stage), **D**egree of **P**arallelism (**DP**, ratio of parallel to total tokens), **A**verage **B**ranch **N**um (**ABN**, average parallel branch num), **P**roportion of **P**arallel **D**ata (**PPD**).



(a) MT Bench. (b) Vicuna Bench. (c) RAG Bench.

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, and the efficiency metrics results are shown in Appendix. 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 Appendix). 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.

Scalability under Batched Serving Following the established evaluation protocol in parallel decoding research, we use single-sequence decoding (i.e., batch size 1) as the primary setting for algorithm-level comparisons. This setting is also adopted by representative methods such as PASTA(Jin et al., 2025), APAR(Liu et al., 2024) and Multiverse(Yang et al., 2025b), as it isolates the acceleration introduced by the decoding algorithm from system-level factors such as batching and request scheduling. To further examine ASPD's scalability in practical serving scenarios, we additionally evaluate it with SGLang under different batch sizes, as reported in Table 2. As the batch size increases from 1 to 128, the wall-clock time decreases substantially from 432.36 seconds to 39.23 seconds, corresponding to an 11.02× speedup and a 90.92% reduction. Notably, ASPD achieves a 7.35× speedup at a batch size of 32, which further increases to 9.52× and 11.02× at batch sizes of 64 and 128, respectively. These results demonstrate that ASPD remains effective under batched serving and scales favorably as the batch size increases. Meanwhile, the marginal gain gradually tapers from batch size 64 to 128, indicating that the system is approaching its computational or hardware utilization limit.

Table 2: Batched serving performance of ASPD with SGLang. Speedup and time reduction are calculated relative to batch size 1.

Batch Size	Wall-Clock Time (s) ↓	Speedup ↑	Time Reduction ↑
1	432.36	1.00×	-
8	137.52	3.14×	68.19%
32	58.79	7.35×	86.40%
64	45.39	9.52×	89.50%
128	39.23	11.02×	90.92%

Table 3: 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.

Data Pipeline			Attention Mask				Position Id		
Method	**Score**	**TPS**	**PosId**	**Attn**	**Score**	**TPS**	**PosId**	**Score**	**TPS**
Baseline	<u>6.21</u>	53.19	Seq	Shared	4.64	**110.30**	Predict	6.75	59.11
APAR*	5.81	59.25	**Seq**	**Indep**	**7.64**	<u>104.21</u>	Same-Max	6.78	89.45
PASTA†	4.98	**106.83**	Max	Shared	3.70	86.96	Same-Re	<u>7.29</u>	<u>95.24</u>
ASPD	**7.64**	<u>104.21</u>	Max	Indep	<u>6.78</u>	89.45	**Same-Seq**	**7.64**	**104.21**

4.3 Ablation Study

4.3.1 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 3 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.

4.3.2 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 3, 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 3). 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.

4.3.3 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 4.3.2. *Same-Re* (Same-Rearrange) reorders position ids sequentially between parallel branches. *Same-Seq* (Same-Sequential) equals to **Seq** in Section 4.3.2.

As evidenced in position id ablation of Table 3, 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 **A**daptive **S**erial-**P**aralle **D**ecoding 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 general tasks and retrieval-augmented generation benchmarks. 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.

AI USE STATEMENT

Generative AI models were used as components of the research methodology for constructing parallelized training data. Specifically, they were used to rewrite serial responses into candidate parallel structures, assess the independence of parallel branches, and verify structural integrity and answer correctness, as described in Section 3.1. Qwen3-235B-A22B was also used as an LLM judge in the evaluation and to improve the training data of the APAR comparison baseline. Generative AI tools, including OpenAI Codex, were additionally used to assist with literature review, manuscript language editing, and checking the paper for consistency with the submission requirements. Model-generated training artifacts were subjected to the filtering and verification procedures described in the paper, and the authors reviewed all AI-assisted analyses and manuscript edits. The authors take full responsibility for the final content of this work, including all text, claims, data, experimental artifacts, and conclusions produced with the assistance of generative AI.

REFERENCES

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, and Tri Dao. Medusa: simple framework for accelerating llm generation with multiple decoding heads (2023). *URL https://github.com/FasterDecoding/Medusa. GitHub repository*, 2023.

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

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

Wei-Lin Chiang, Zhuohan Li, Ziqing Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Chan-Jan Hsu, Davide Buffelli, Jamie McGowan, Feng-Ting Liao, Yi-Chang Chen, Sattar Vakili, and Da-shan Shiu. Group think: Multiple concurrent reasoning agents collaborating at token level granularity. *arXiv preprint arXiv:2505.11107*, 2025.

Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL `https://github.com/huggingface/open-r1`.

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

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

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025.

Mingdao Liu, Aohan Zeng, Bowen Wang, Peng Zhang, Jie Tang, and Yuxiao Dong. Apar: Llms can do auto-parallel auto-regressive decoding. *arXiv preprint arXiv:2401.06761*, 2024.

LMSYS. Sharegpt vicuna unfiltered, 2023. URL https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered.

Neural Bridge AI. Retrieval-augmented generation (rag) dataset 12000, 2023. URL https://huggingface.co/datasets/neural-bridge/rag-dataset-12000.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-thought: Prompting llms for efficient parallel generation. *arXiv preprint arXiv:2307.15337*, 2023.

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

Qwen Team. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

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

ByteDance Seed, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, et al. Seed1. 5-thinking: Advancing superb reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.13914*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Sam Wiseman and Alexander M Rush. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*, 2016.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Hanling Yi, Feng Lin, Hongbin Li, Peiyang Ning, Xiaotian Yu, and Rong Xiao. Generation meets verification: Accelerating large language model inference with smart parallel auto-correct decoding. *arXiv preprint arXiv:2402.11809*, 2024.

Yijiong Yu. Accelerate parallelizable reasoning via parallel decoding within one sequence. *arXiv preprint arXiv:2503.20533*, 2025.

Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint arXiv:2406.07394*, 2024a.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint arXiv:2309.08168*, 2023.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. *arXiv preprint arXiv:2408.15766*, 2024b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37: 62557–62583, 2024.

A Appendix

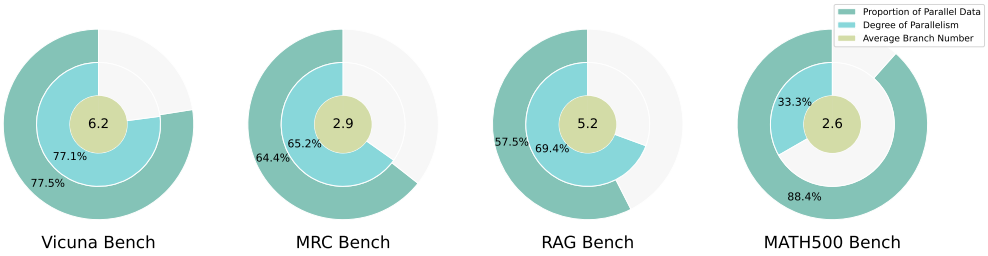


Figure 5: Parallelization patterns in our trained parallel model's responses across different scenarios.

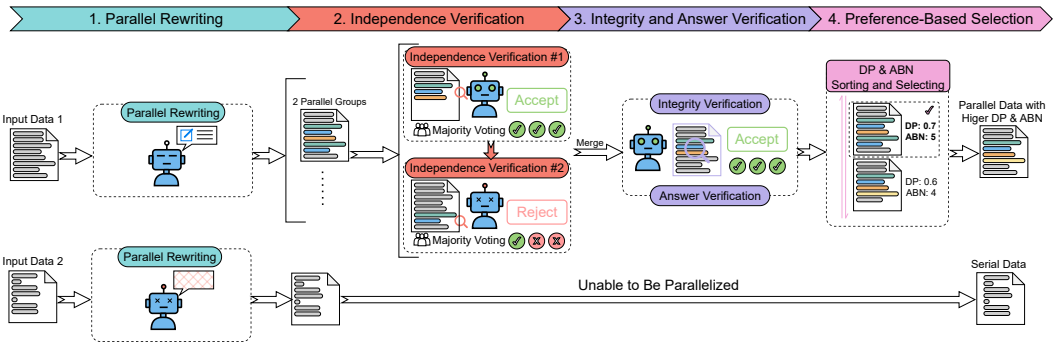


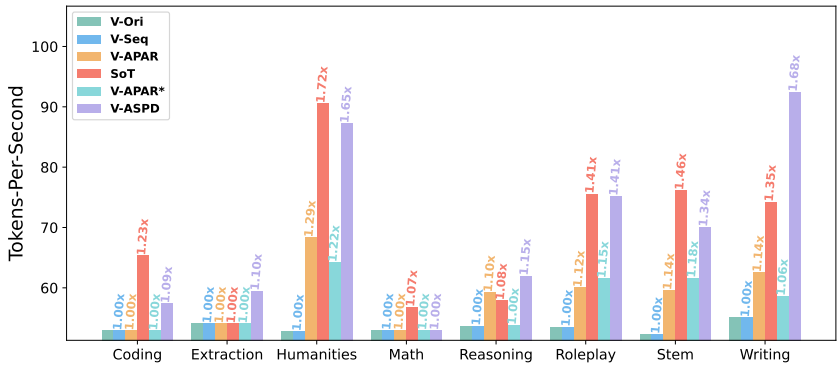
Figure 6: **Non-Invasive Parallel Data Transformation Pipeline.**

A.1 Full Performance Comparison on MT Bench and Vicuna Bench

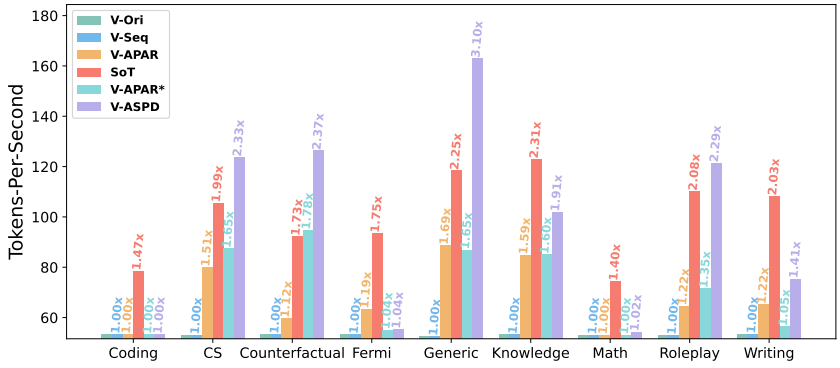
Table 4 shows complete experimental results of various subtasks on MT Bench and Vicuna Bench, and the corresponding acceleration ratio of each subtask is presented in Figure 7.

A.2 Wall Clock Latency Speedup

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

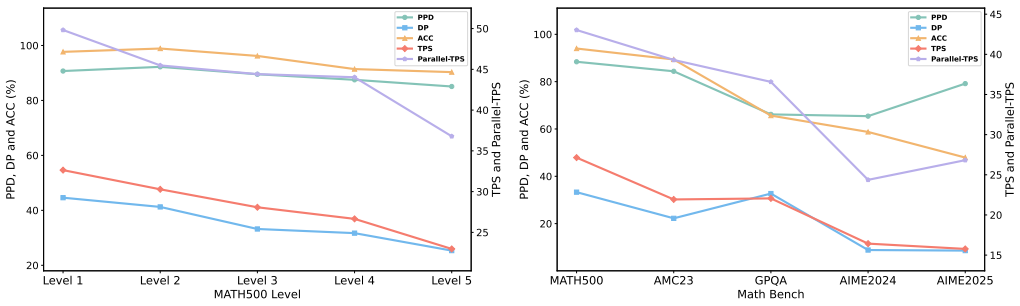


(a) MT Bench.



(b) Vicuna Bench.

Figure 7: 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 8: Analysis between Parallelization Metrics and Problem Difficulty.

Table 4: Performance comparison on MT Bench and Vicuna Bench. CS is short for Common-Sense on Vicuna Bench.

MT Bench									
Task \ Model	V-Ori	V-Seq	V-APAR	SoT	V-APAR*	**V-ASPD**	Q-Ori	Q-Seq	**Q-ASPD**
Coding	**3.50**	3.10	2.70	3.00	2.55	2.90	**7.15**	6.60	6.70
Extraction	5.10	4.40	**5.20**	3.08	5.05	4.80	7.45	6.93	**7.80**
Humanities	6.55	**8.40**	6.40	6.30	8.10	7.95	8.70	**9.15**	9.05
Math	2.95	**3.05**	2.55	2.75	2.50	3.00	8.50	8.45	**8.90**
Reasoning	4.65	4.80	5.45	4.50	4.75	**5.85**	6.85	**7.25**	7.05
Roleplay	5.30	7.10	5.60	5.60	6.80	**7.25**	8.10	8.40	**8.48**
Stem	5.55	6.20	5.55	5.85	**6.50**	6.30	7.80	**8.60**	8.50
Writing	5.30	**7.65**	5.60	4.75	6.80	6.70	8.00	8.50	**8.75**
Mean	4.86	**5.59**	4.88	4.48	5.38	**5.59**	7.82	7.98	**8.15**
Vicuna Bench									
Task \ Model	V-Ori	V-Seq	V-APAR	SoT	V-APAR*	**V-ASPD**	Q-Ori	Q-Seq	**Q-ASPD**
Coding	**4.29**	3.71	4.14	3.43	3.71	3.71	**9.00**	8.29	8.71
CS	7.40	**9.00**	7.20	7.30	8.90	8.90	9.00	**9.30**	9.00
Counterfactual	5.10	8.50	5.30	5.30	8.40	**8.60**	8.40	**9.10**	**9.10**
Fermi	**5.50**	4.50	4.60	4.40	5.00	5.20	7.70	**8.20**	8.10
Generic	7.30	**9.20**	7.40	7.40	8.95	8.80	8.70	**9.55**	9.40
Knowledge	7.30	**9.10**	7.20	7.50	8.60	9.00	9.00	**9.40**	9.10
Math	2.67	2.33	**3.33**	2.00	2.33	2.67	9.67	**10.00**	**10.00**
Roleplay	7.20	**9.20**	6.60	6.70	9.00	9.00	9.00	9.30	**9.40**
Writing	6.10	8.80	6.60	5.80	8.80	**9.00**	8.20	**9.20**	9.00
Mean	6.21	7.70	6.10	5.93	7.62	**7.74**	8.65	**9.11**	9.03

Building on our analysis of parallelization potential, 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. 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.3 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,

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

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 7, 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 6. 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.

A.4 In-depth Analysis of Mathematical Results

Parallelism Variation Across Tasks Compared to the results of other scenarios, 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 (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 the main paper, 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 8a clearly shows that parallel efficiency metrics decrease as problem difficulty increases. This inverse relationship is consistently observed across various mathematical benchmarks in Figure 8b, where higher difficulty levels corresponds to 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.

A.5 Limitations and Future Work

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

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

Table 6: Comprehensive parallelization metrics on our internal test sets.

Test Set	PPD	DP	ABN	P-TPS
MRC	64.40	65.21	2.86	$109.32_{2.18x}$
LR	44.59	55.80	2.40	$101.72_{1.92x}$
AI Search	64.50	67.69	3.21	$97.27_{2.41x}$

Table 7: 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	<u>40.29</u>
Q-Seq	**73.20**	<u>50.53</u>	**56.69**	<u>54.32</u>	**8.23**	39.89
Q-ASPD	<u>72.40</u>	**67.91**	<u>55.41</u>	**61.26**	<u>8.15</u>	**58.56**

Within our parallel data production pipeline, we have established several key evaluation metrics, including TPS, DP, and ABN. These quantitative indicators demonstrate significant potential for integration into reinforcement learning frameworks, where they could effectively guide model optimization towards achieving enhanced parallelism while maintaining or improving overall performance.