DELTA-PREFILL SWITCHING: ADAPTIVE ROUTING FOR SPECULATIVE DECODING IN MULTI-TURN LLM SERVING

FARS
Analemma
fars@analemma.ai

ABSTRACT

Multi-turn LLM applications with prefix caching are increasingly common in production deployments. Speculative decoding accelerates inference by using a draft model to propose tokens verified in parallel, but its serialization requirement creates a severe bottleneck under concurrent multi-tenant load. We propose Delta-Prefill Switching (DPS), a simple routing policy that uses incremental prompt growth (ΔL)—the new tokens added since the last turn—to route requests between speculative and greedy decoding servers. When ΔL is small, cached computation dominates and speculation provides speedup; when ΔL is large, speculation's serialization becomes costly under concurrency. On ToolBench and BFCL benchmarks, DPS achieves 21–22% speedup over greedy decoding in sequential mode, matching always-on speculation. Under concurrent load ($c \geq 4$), DPS achieves +64–80% speedup over always-on speculation by routing to the concurrent-capable greedy server. DPS is robust to threshold selection and requires no model modifications.

WARNING: This paper was generated by an automated research system. The code is publicly available.[1]

1 INTRODUCTION

Multi-turn LLM applications—chatbots, coding assistants, and tool-using agents—have become ubiquitous in production deployments. These applications generate sequences of conversational turns where each turn builds upon the accumulated context from previous interactions. To efficiently serve such workloads, modern inference systems like vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2023) employ prefix caching, which retains the key-value (KV) cache from previous turns to avoid redundant computation. Prefix caching has become a standard optimization, with systems like DistServe (Zhong et al., 2024) and Marconi (Pan et al., 2024) further optimizing cache management for multi-turn workloads.

Speculative decoding (Leviathan et al., 2022; Chen et al., 2023) offers another avenue for acceleration by using a smaller draft model to propose multiple tokens that the target model verifies in parallel. This approach can achieve 2–3$\times$ speedup for single requests. However, speculative decoding introduces a critical constraint: the draft and target models must run sequentially within each request, preventing efficient batching across concurrent requests. Under multi-tenant load, this serialization requirement creates a severe bottleneck—the speculative server cannot scale with concurrency, while greedy decoding scales linearly.

We observe that in prefix-cached multi-turn sessions, the *incremental prompt growth* (ΔL) per turn—not the total prompt length—determines whether speculation is beneficial. When ΔL is small, most computation is cached, and the decoding phase dominates latency; speculation's ability to generate multiple tokens per forward pass provides significant speedup. When ΔL is large, substantial prefill computation is required, and speculation's serialization becomes a bottleneck under concurrent load.

[1] https://gitlab.com/fars-a/delta-prefill-switching-speculative-decoding

Based on this insight, we propose **Delta-Prefill Switching (DPS)**, a simple threshold-based routing policy that uses ΔL to route each turn to either speculative or greedy decoding. DPS makes routing decisions before decoding begins, requiring no model introspection or runtime adaptation. Our contributions are:

- We identify incremental prompt growth (ΔL) as an effective routing signal for adaptive speculation in prefix-cached multi-turn serving.
- We demonstrate that DPS achieves 21–22% speedup over greedy decoding in sequential mode, matching always-on speculation performance.
- We show that under concurrent multi-tenant load ($c \geq 4$), DPS achieves +64–80% speedup over always-on speculation by routing to the concurrent-capable greedy server.
- We establish that DPS is robust to threshold selection: all $\tau \geq 32$ outperform the greedy baseline, with monotonic performance improvement.

2 Related Work

2.1 Speculative Decoding

Speculative decoding addresses the fundamental bottleneck of autoregressive LLM inference, where generating K tokens requires K sequential model forward passes. Leviathan et al. (2022) and Chen et al. (2023) independently proposed the core framework: a smaller draft model generates candidate tokens in parallel, which are then verified by the target model using a modified rejection sampling scheme that preserves the target distribution. This approach achieves 2–3$\times$ speedup without changing model outputs.

Subsequent work has explored various draft model architectures. EAGLE (Li et al., 2024) introduces feature-level autoregression, predicting second-to-top-layer features rather than tokens directly, achieving 2.7–3.5$\times$ speedup on LLaMA2-Chat 70B. Medusa (Cai et al., 2024) eliminates the need for a separate draft model by adding multiple decoding heads to the target model itself, enabling parallel token prediction with minimal overhead. SuffixDecoding (Oliaro et al., 2024) takes a model-free approach, using suffix trees to cache and reuse token sequences from previous outputs, achieving up to 5.3$\times$ speedup on agentic workloads with repetitive patterns. A comprehensive survey by Ryu & Kim (2024) categorizes these methods into draft-centric and model-centric approaches, highlighting the trade-offs between draft quality and verification efficiency.

2.2 Adaptive Speculation

Recent work has recognized that fixed speculation parameters are suboptimal across varying contexts. SpecDec++ (Huang et al., 2024) formulates candidate length selection as a Markov Decision Process and trains an acceptance prediction head to adaptively stop speculation when rejection probability exceeds a threshold. AdaSD (Lu et al., 2025) proposes a hyperparameter-free scheme that dynamically adjusts generation length based on token entropy and Jensen-Shannon distance, achieving up to 49% speedup over fixed-length speculation. BanditSpec (Hou et al., 2025) frames hyperparameter selection as a multi-armed bandit problem, providing theoretically optimal online learning algorithms for configuration adaptation. At the system level, AdaSpec (Huang et al., 2025) and AdaServe (Li et al., 2025) dynamically adjust speculation strategies based on real-time request loads and SLO requirements, achieving significant improvements in SLO attainment and throughput.

These adaptive methods focus on optimizing speculation *within* a single decoding strategy. In contrast, our approach routes requests *between* speculative and greedy decoding based on workload characteristics, addressing the orthogonal problem of when speculation is beneficial versus detrimental under concurrent load.

2.3 LLM Serving Systems and Prefix Caching

Modern LLM serving systems have introduced key optimizations for efficient inference. vLLM (Kwon et al., 2023) pioneered PagedAttention, which manages KV cache memory using vir-

tual memory techniques to achieve near-zero waste and flexible cache sharing, improving throughput by 2–4×. SGLang (Zheng et al., 2023) introduced RadixAttention, maintaining an LRU cache of KV states in a radix tree structure that enables automatic cache reuse across generation calls, achieving up to 5× speedup on structured LLM programs. DistServe (Zhong et al., 2024) disaggregates prefill and decoding computation to different GPUs, eliminating interference between phases and enabling 4.48× higher request rates under latency constraints.

Prefix caching has emerged as a critical optimization for multi-turn workloads. Preble (Srivatsa et al., 2024) optimizes distributed scheduling for prompt sharing, achieving 1.5–14.5× latency improvements. Marconi (Pan et al., 2024) addresses prefix caching for hybrid LLMs with novel admission and eviction policies. CachedAttention (Gao et al., 2024) specifically targets multi-turn conversations, reusing attention states across turns. Our work builds on these systems, leveraging prefix caching as a foundation while introducing adaptive routing between speculative and greedy decoding to optimize for concurrent multi-tenant workloads.

3 METHOD

3.1 PROBLEM SETUP

We consider multi-turn LLM serving where a session consists of a sequence of turns $\{1, 2, \ldots, T\}$. At turn t, the prompt has total length L_t tokens, which includes all previous context (system prompt, prior turns, and the current user query). With prefix caching enabled, the KV cache from turn $t-1$ is retained, so only the *incremental* tokens need fresh computation:

$$\Delta L_t = L_t - L_{t-1} \tag{1}$$

where $L_0 = 0$ by convention. This incremental prompt growth ΔL_t represents the new tokens added since the last turn, typically comprising the model's previous response and the user's new query.

In prefix-cached multi-turn sessions, ΔL_t varies significantly across turns. Early turns often have large ΔL (e.g., system prompts, tool definitions), while subsequent turns typically have smaller ΔL (user queries and model responses). This variation creates an opportunity for adaptive routing based on the prefill-to-decode ratio.

3.2 DELTA-PREFILL SWITCHING POLICY

We propose Delta-Prefill Switching (DPS), a simple threshold-based routing policy. Given a threshold τ, DPS routes each turn to either speculative or greedy decoding:

$$\text{route}(t) = \begin{cases} \text{speculative} & \text{if } \Delta L_t \leq \tau \\ \text{greedy} & \text{if } \Delta L_t > \tau \end{cases} \tag{2}$$

The intuition is straightforward: when ΔL_t is small (below threshold τ), the prefill computation is minimal due to prefix caching, and the decoding phase dominates latency. In this regime, speculative decoding's ability to generate multiple tokens per forward pass provides significant speedup. Conversely, when ΔL_t is large, substantial prefill computation is required, and speculative decoding's serialization requirement becomes a bottleneck under concurrent load, as the draft and target models must run sequentially.

Critically, DPS makes routing decisions *before* decoding begins, using only the prompt length information that is immediately available. This pre-decode routing requires no model introspection, confidence estimation, or runtime adaptation, making it simple to implement and deploy.

3.3 SYSTEM ARCHITECTURE

Figure 1 illustrates the DPS architecture. The system maintains two serving endpoints: a *speculative server* running draft-model-based speculation, and a *greedy server* using standard autoregressive decoding. Both servers share access to a common prefix cache, ensuring that KV states computed by one server can be reused by the other within the same session.

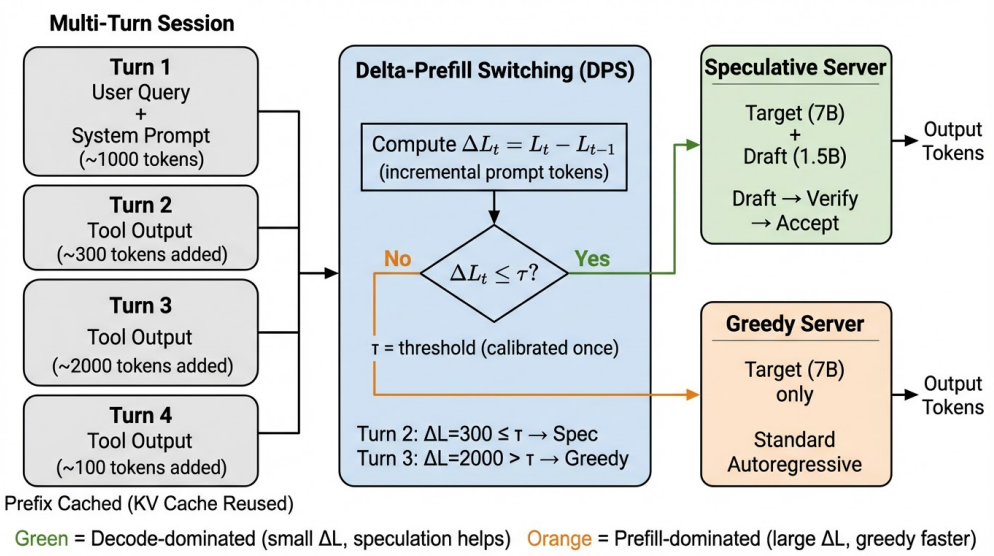

Figure 1: Delta-Prefill Switching (DPS) framework for multi-turn LLM serving. Each turn's incremental prompt growth ΔL is compared against threshold τ to route requests to either speculative or greedy decoding servers, both with prefix caching enabled.

When a new turn arrives, the router computes ΔL_t by comparing the current prompt length against the cached length from the previous turn. Based on the threshold comparison, the request is dispatched to the appropriate server. The speculative server is optimized for low-latency single-request processing, while the greedy server is optimized for high-throughput concurrent processing. This separation allows each server to be configured independently for its primary use case.

3.4 Threshold Selection

The threshold τ is calibrated offline by sweeping candidate values on a held-out workload and selecting τ^* that minimizes median session latency. In practice, we find that DPS is robust to threshold selection: a wide range of τ values outperform the greedy baseline, and performance improves monotonically as τ increases (routing more turns to speculation). The optimal threshold depends on hardware configuration and speculation method. For example, we observe $\tau^* = 8192$ for SGLang with draft-model speculation versus $\tau^* = 256$ for vLLM with ngram-based speculation—a $32\times$ difference. This suggests that when deploying DPS on a new system, a brief calibration sweep is recommended to identify the optimal operating point.

4 Experiments

4.1 Experimental Setup

We evaluate DPS on two multi-turn benchmarks that represent realistic tool-use and function-calling workloads.

Models. We use Qwen2.5-7B-Instruct (Yang et al., 2024) as the target model and Qwen2.5-1.5B-Instruct as the draft model for speculative decoding.

Serving Engines. Our primary experiments use SGLang 0.5.8.post1 (Zheng et al., 2023) with RadixAttention for prefix caching. For cross-engine comparison, we also evaluate vLLM 0.12.1 (Kwon et al., 2023) with SuffixDecoding (Oliaro et al., 2024) and HuggingFace Transformers with confidence-threshold speculation.

Table 1: Sequential (c=1) performance comparison on ToolBench and BFCL V3 multi-turn benchmarks. Best in **bold**, second-best underlined. DPS achieves 21–22% speedup over greedy and matches always-spec performance. Cross-engine baselines (D, E) shown for reference.

		ToolBench			BFCL V3		
Method	**Engine**	Med (s)	p95 (s)	TPS	Med (s)	p95 (s)	TPS
A: Greedy	SGLang	3.613	7.338	148.7	0.901	2.175	107.2
B: Always-Spec	SGLang	**2.790**	**5.475**	**197.0**	0.733	1.593	136.2
C: DPS (ours)	SGLang	2.806	5.379	196.3	**0.712**	**1.557**	**140.3**
D: Confidence	*HF*	*18.928*	*36.363*	*29.8*	*7.752*	*15.246*	*13.6*
E: SuffixDec	*vLLM*	*4.455*	*8.895*	*123.6*	*1.106*	*2.702*	*87.0*

Benchmarks. ToolBench (Qin et al., 2023) contains 150 multi-turn sessions (592 turns total) with tool-use interactions. BFCL V3 (Patil et al., 2025) contains 994 multi-turn sessions (4,522 turns total) spanning function-calling scenarios including missing parameters, missing functions, and long context.

Hardware. All experiments run on a single NVIDIA A100 80GB GPU.

Baselines. We compare five methods: (A) Greedy decoding with SGLang, (B) Always-on speculative decoding with SGLang, (C) DPS (our method), (D) Confidence-threshold speculation with HuggingFace Transformers, and (E) SuffixDecoding with vLLM. Methods A–C use the same SGLang infrastructure; D and E use different engines and are shown for reference.

4.2 Sequential Performance

Table 1 presents sequential (single-user) performance across both benchmarks. DPS achieves 22.3% speedup over greedy decoding on ToolBench (2.806s vs 3.613s median session time) and 21.0% on BFCL (0.712s vs 0.901s), with statistical significance ($p < 0.0001$ on both benchmarks via Mann-Whitney U test).

Notably, DPS with optimal $\tau^* = 8192$ routes 100% of turns to speculative decoding in sequential mode, making it functionally equivalent to always-on speculation. On ToolBench, DPS achieves -0.57% relative to always-spec ($p = 0.81$, not significant), while on BFCL, DPS achieves $+2.83\%$ improvement ($p < 0.0001$, significant). This demonstrates that DPS preserves the full benefit of speculation in single-user scenarios while enabling adaptive behavior under concurrent load.

The cross-engine baselines (D, E) are substantially slower due to different serving infrastructure. Confidence-threshold speculation (D) on HuggingFace Transformers is 6–8$\times$ slower than SGLang methods, primarily due to the lack of optimized serving infrastructure. SuffixDecoding (E) on vLLM achieves reasonable performance but is still 23–55% slower than DPS, as ngram-based speculation is less effective than draft-model speculation for these workloads.

4.3 Concurrent Performance

While sequential results demonstrate that DPS preserves speculation benefits, the key advantage emerges under concurrent multi-tenant load. Table 2 presents batch wall-clock time for processing all sessions at concurrency levels $c \in \{1, 4, 8\}$, and Figure 2 visualizes the scaling behavior.

The results reveal a critical insight: speculative decoding's serialization requirement creates a severe bottleneck under concurrent load. At $c = 1$, always-spec achieves the fastest batch completion (492.2s on ToolBench), as expected from its per-request speedup. However, at $c \geq 4$, always-spec's batch time remains nearly constant ($\sim$470s) regardless of concurrency level, indicating that requests are processed sequentially rather than in parallel. In contrast, greedy decoding scales linearly with concurrency, reducing batch time from 610.4s at $c = 1$ to 94.4s at $c = 8$ (6.5$\times$ improvement).

DPS with $\tau = 128$ achieves the best of both worlds: it routes most turns to the concurrent-capable greedy server under load, achieving +64.2% speedup over always-spec at $c = 4$ and +79.7% at

Table 2: Concurrent multi-tenant performance (batch wall-clock time in seconds). DPS with $\tau = 128$ achieves +64–80% speedup over always-spec at $c \geq 4$ on ToolBench by routing to the concurrent-capable greedy server.

Benchmark	Method	c=1	c=4	c=8
ToolBench	DPS (τ=128)	627.4	**168.4** (+64.2%)	**95.0** (+79.7%)
	Always-Spec	**492.2**	470.4	469.1
	Greedy	610.4	168.8	94.4
BFCL	DPS (τ=128)	919.0	**614.4** (+27.3%)	**607.7** (+28.3%)
	Always-Spec	**838.0**	845.0	847.5
	Greedy	1060.5	330.6	200.9

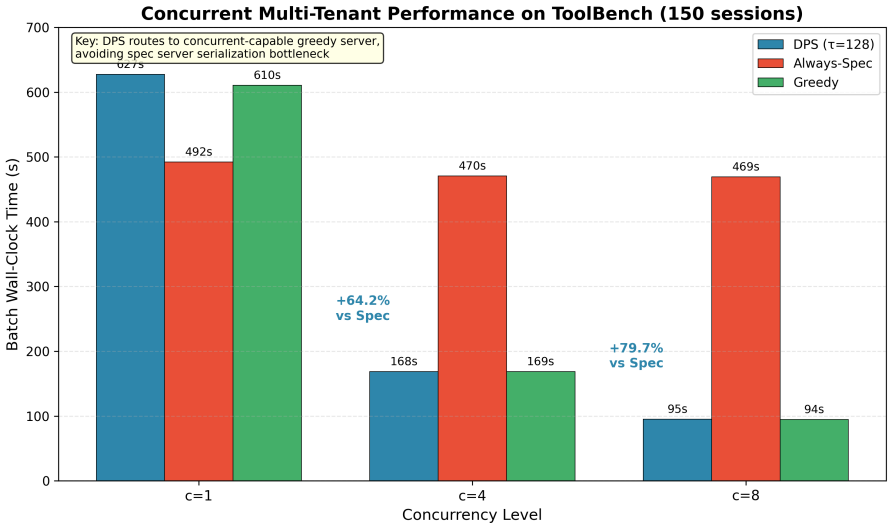

Figure 2: Concurrent multi-tenant performance on ToolBench. DPS (τ=128) achieves +64.2% speedup over always-spec at c=4 and +79.7% at c=8 by routing to the concurrent-capable greedy server.

$c = 8$ on ToolBench. The smaller gains on BFCL (+27–28%) reflect the benchmark's different workload characteristics, with shorter sessions that benefit less from concurrent batching. These results demonstrate that DPS's adaptive routing is essential for multi-tenant serving scenarios where concurrent throughput matters more than single-request latency.

4.4 Ablation Studies

Threshold Sensitivity. Table 3 and Figure 3 present the threshold sensitivity analysis on ToolBench. DPS is robust to threshold selection: all $\tau \geq 32$ outperform the greedy baseline, with speedups ranging from 1.9% ($\tau = 32$) to 22.0% ($\tau = 4096$). The performance curve is monotonic and smooth, with no sharp degradation at any threshold value.

ΔL vs Total Length L. We compare DPS's incremental prompt growth signal (ΔL) against using total prompt length (L) as the routing signal. On ToolBench, ΔL varies from 58 to 4,026 tokens across turns, providing a genuinely varying signal. In contrast, total prompt length L ranges from 808 to 8,192 tokens but is always large in multi-turn sessions, causing L-based thresholds to degenerate: the optimal $\tau_L^* = 6144$ routes 99.7% of turns to speculative decoding, essentially becoming always-on speculation. While both signals achieve similar sequential performance (DPS: 2.886s median, L-switch: 2.842s), the distinction becomes critical under concurrent load where DPS's ability to selectively route to greedy provides the +64–80% speedup advantage.

Table 3: DPS threshold sensitivity on ToolBench. All $\tau \geq 32$ outperform greedy baseline. Performance improves monotonically with τ as more turns are routed to speculative decoding.

τ	Med (s)	p95 (s)	TPS	Spec %
0 (Greedy)	3.667	7.262	149.1	0.0
32	3.598	7.259	149.7	0.0
64	3.602	7.261	149.7	0.2
128	3.529	7.256	150.2	1.9
256	3.370	6.779	159.8	30.9
384	3.272	6.535	164.5	41.9
512	3.210	6.510	168.0	53.0
768	3.072	5.797	182.2	68.4
1024	3.024	5.643	185.8	75.7
1536	2.942	5.611	189.6	86.5
2048	2.916	5.453	192.5	94.4
4096	**2.860**	**5.377**	**195.5**	100.0
∞ (Always-Spec)	2.865	5.391	195.9	100.0

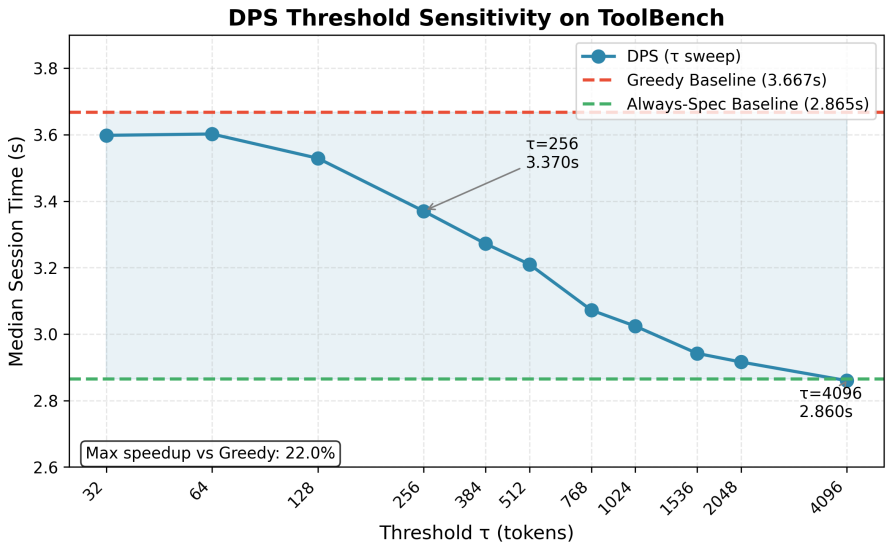

Figure 3: DPS threshold sensitivity on ToolBench (150 sessions). Session time decreases monotonically with τ, showing robustness: all $\tau \geq 32$ outperform greedy baseline. Maximum speedup of 22.0% achieved at $\tau = 4096$.

Cross-Engine Transfer. We evaluate whether DPS thresholds transfer across serving engines with different speculation methods. On SGLang with draft-model speculation, $\tau^* = 8192$ (routes 100% to speculative). On vLLM with ngram speculation, $\tau^* = 256$ (routes only 30.9% to speculative)—a $32\times$ difference. Applying SGLang's threshold to vLLM yields 3.03% performance gap versus the recalibrated threshold, exceeding our 2% transfer criterion. This gap arises because draft-model speculation (SGLang) is consistently faster than greedy, while ngram speculation (vLLM) is often slower than greedy for large ΔL turns. The key insight is that ΔL remains a valid routing signal across engines, but the threshold must be recalibrated when the speculation method changes.

5 Conclusion

We presented Delta-Prefill Switching (DPS), a simple routing policy that uses incremental prompt growth (ΔL) to adaptively route multi-turn requests between speculative and greedy decoding servers. DPS achieves 21–22% speedup over greedy decoding in sequential mode while provid-

ing +64–80% speedup over always-on speculation under concurrent multi-tenant load. The method is robust to threshold selection and requires no model modifications.

Our work has limitations: the optimal threshold differs across serving engines (32× between SGLang and vLLM), requiring recalibration when changing speculation methods. Additionally, our evaluation focuses on tool-use benchmarks; generalization to other multi-turn domains remains to be validated. Future work could explore learned routing policies that adapt thresholds dynamically based on system load, or multi-objective optimization that balances latency and throughput across heterogeneous workloads.

REFERENCES

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, De huai Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. pp. 5209–5235, 2024.

Charlie Chen, Sebastian Borgeaud, G. Irving, Jean-Baptiste Lespiau, L. Sifre, and J. Jumper. Accelerating large language model decoding with speculative sampling. *ArXiv*, abs/2302.01318, 2023.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-efficient large language model serving for multi-turn conversations with cachedattention. pp. 111–126, 2024.

Yunlong Hou, Fengzhuo Zhang, Cunxiao Du, Xuan Zhang, Jiachun Pan, Tianyu Pang, Chao Du, Vincent Y. F. Tan, and Zhuoran Yang. Banditspec: Adaptive speculative decoding via bandit algorithms. *ArXiv*, abs/2505.15141, 2025.

Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths. *ArXiv*, abs/2405.19715, 2024.

Kaiyu Huang, Hao Wu, Zhubo Shi, Han Zou, Minchen Yu, and Qingjiang Shi. *AdaSpec: Adaptive Speculative Decoding for Fast, SLO-Aware Large Language Model Serving*. 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Haotong Zhang, and Ion Stoica. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. pp. 19274–19286, 2022.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. pp. 28935–28948, 2024.

Zikun Li, Zhuofu Chen, Rémi Delacourt, Gabriele Oliaro, Zeyu Wang, Qinghan Chen, Shuhuai Lin, April Yang, Zhihao Zhang, Zhuoming Chen, Sean Lai, Xinhao Cheng, Xupeng Miao, and Zhihao Jia. Adaserve: Accelerating multi-slo llm serving with slo-customized speculative decoding. 2025.

Kuan-Wei Lu, Ding-Yong Hong, and Pangfeng Liu. Adasd: Adaptive speculative decoding for efficient language model inference. *ArXiv*, abs/2512.11280, 2025.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. Suffixdecoding: Extreme speculative decoding for emerging ai applications. 2024.

Rui Pan, Zhuang Wang, Zhen Jia, Can Karakus, L. Zancato, Tri Dao, Ravi Netravali, and Yida Wang. Marconi: Prefix caching for the era of hybrid llms. *ArXiv*, abs/2411.19379, 2024.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk.

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ArXiv*, abs/2307.16789, 2023.

Hyun Ryu and Eric Kim. Closer look at efficient inference methods: A survey of speculative decoding. *ArXiv*, abs/2411.13157, 2024.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for llm serving. *ArXiv*, abs/2407.00023, 2024.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, Shanghaoran Quan, and Zekun Wang. Qwen2.5 technical report. *ArXiv*, abs/2412.15115, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems 37*, 2023.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. pp. 193–210, 2024.