

Enterprise Latency Optimization, Streaming Infrastructure, and Retrieval-Augmented Generation Architectures: Research Innovations in Enterprise Conversational AI Performance

Prasun Bhattacharyya

Abstract: Enterprise conversational AI platforms now serve millions of concurrent users, and latency has become a structural design constraint rather than a post-launch tuning target. Most published optimization research addresses a narrow slice of this problem: how quickly a single language model call completes, through techniques such as continuous batching, paged memory management, and speculative decoding. This paper argues that orchestration and routing overhead, not model inference itself, is the dominant addressable latency bottleneck in production enterprise conversational AI, and that the field's heavy concentration on model-level techniques is somewhat misallocated relative to the parallelism available at the pipeline level. We introduce a pipeline decomposition formula, $TM99 = \text{WebSocket} + \max(\text{LLM Routing}, \text{Contextualization} + \max(\text{Cache Lookup}, \text{Guardrail})) + \text{Plugin Execution} + \text{Response Synthesis}$, that expresses end-to-end tail latency as a function of sequential and parallelizable stages, and we describe the three-path execution model used to realize that parallelism in production. Drawing on the caching, routing, and speculative-execution literature, we survey the measured gains reported there, ranging from single-digit-percentage improvements to more than sixty-fold reductions in time to first token depending on technique and cache hit rate, and we argue for TM50/TM90/TM99 percentile benchmarking over mean-latency reporting, which obscures the tail behavior that governs service-level compliance. The analysis indicates that orchestration-layer parallelism frequently exceeds available model-level gains in enterprise multi-agent settings, and we outline where practitioners should concentrate optimization effort accordingly.

Keywords: latency optimization; retrieval-augmented generation; streaming infrastructure; semantic caching; percentile benchmarking; pipeline orchestration

1. Introduction

Enterprise conversational AI has moved from a novelty feature to core infrastructure. Millions of concurrent users interact daily with AI assistants embedded in customer service platforms, internal tooling, and workflow automation across large organizations. The transformer architecture underlying nearly all modern large language models made this shift possible [1], and retrieval-augmented generation extended these models' usefulness by grounding responses in external knowledge sources [2]. Both innovations solved a capability problem: could a model produce accurate, contextually relevant answers. Neither was designed to solve a systems problem: could that answer arrive fast enough to sustain a real-time conversation at scale.

Latency in this setting is not a cosmetic concern. When response times exceed a few hundred

Sr. Technical Program Manager, Enterprise AI Platforms (Generative AI), Amazon, USA

ORCID ID: 0009-0005-6691-6809

milliseconds to first token, users perceive lag, abandon tasks, or route around the assistant entirely. At enterprise scale, that abandonment translates into lost productivity, degraded customer experience, and quantifiable revenue loss. A platform serving millions of concurrent sessions cannot treat latency as something to fix after launch. It has to be an architectural constraint present from the first design decision, on par with availability or data security.

Yet the bulk of published optimization research targets a narrower question: how quickly can a single model generate tokens. Quantization, distillation, efficient attention handling, and inference-serving improvements have delivered substantial, well-documented gains at the model-serving layer, and the section that follows reviews that work in detail. A production conversational AI request, however, rarely touches only a model. It passes through a WebSocket connection, a routing decision that selects among model tiers, a contextualization step that retrieves relevant

documents, a guardrail check that screens for safety and compliance violations, one or more plugin calls that fetch external data or execute actions, and a response synthesis stage that assembles and streams the final answer. Each stage adds latency, and several of them run in sequence today when they could run in parallel.

This raises the structural question this paper investigates: is the field's optimization effort aimed at the part of the pipeline that actually determines end-to-end latency in production? Several years spent operating latency-reduction programs across large-scale enterprise AI platforms serving millions of concurrent users point toward no. Model-level inference optimization is necessary but insufficient on its own. Orchestration and routing overhead, not model inference itself, has become the dominant addressable bottleneck in enterprise-grade conversational AI, and this imbalance carries practical consequences for where engineering effort and research attention should go next.

The remainder of this paper develops that argument in four steps. Section 2 reviews the existing latency and retrieval-augmented generation optimization literature and introduces a pipeline decomposition formula that expresses end-to-end latency as a function of both sequential and parallel pipeline stages. Section 3 describes the three-path execution model used in production systems to separate routing, contextualization, and safety-checking work, along with the distinction between first-byte and last-byte latency that streaming architectures are built around. Section 4 evaluates measured gains from prompt caching, model tiering, and semantic caching against percentile-based benchmarking data, and discusses where published results transfer cleanly to multi-agent enterprise deployments and where they do not. Section 5 closes by reframing where optimization effort should concentrate going forward.

2. Literature Review and Pipeline Decomposition

Efficient serving of transformer models has been studied extensively at the systems level. Pope et al. examined how attention computation, memory bandwidth, and batching interact as models scale, establishing much of the vocabulary later serving research relies on [3]. Building on that foundation, Orca introduced iteration-level scheduling for generative model serving and reported a 36.9x throughput improvement at the same level of latency

relative to NVIDIA's FasterTransformer on GPT-3 175B [4], a result that reframed batching itself as a latency lever rather than a purely throughput lever. PagedAttention, implemented in the vLLM serving system, addressed memory fragmentation in the key-value cache and improved throughput of popular large language models by 2-4x at the same level of latency [5]. These papers share a common orientation: they treat the model server as the unit of optimization, and they measure success primarily in throughput terms later translated into latency claims.

A parallel line of work targets the decoding process itself. Speculative decoding generates candidate tokens with a smaller draft model and verifies them with the target model, reporting a 2x-3x wall-clock speedup on T5-XXL with no architectural changes and no retraining [6]. Speculative sampling applies a related technique to Chinchilla 70B and reports a 2-2.5x decoding speedup [7]. GPTCache takes a different approach, matching new queries against previously cached queries and responses through semantic similarity, and reports response speed increases of 2 to 10 times when a cache hit occurs [8]. MLPerf's inference benchmark, drawing on more than 600 reproducible measurements from 14 organizations across more than 30 systems, has become a reference point for comparing serving approaches under standardized conditions [9]. Miao et al. survey this broader landscape and describe it as an algorithms-to-systems continuum spanning decoding algorithms, serving architectures, and hardware-aware scheduling [10].

Several systems push further into architecture-level restructuring. Tabi routes simpler queries to smaller models and reserves larger models for harder queries, reporting a 21%-40% reduction on average latency against comparable inference systems [11]. Attention-sink methods extend context handling in streaming scenarios and report speedups of up to 22.2x over sliding-window recomputation baselines under the conditions tested [12]. DistServe separates the prefill and decoding phases onto different resources, reporting the ability to serve 7.4x more requests or meet 12.6x tighter service-level objectives while keeping more than 90% of requests within latency constraints [13]. Splitwise applies a related phase-splitting principle and reports up to 1.4x higher throughput at 20% lower cost, or 2.35x more throughput under matched power and cost budgets [14].

Retrieval-augmented generation layers its own latency profile on top of model inference. Lewis et al.'s original formulation combines a parametric generator with a non-parametric retrieval index [2], and Self-RAG extends this by teaching the model to decide when retrieval is necessary and to critique its own output [15]. Both improve answer quality; neither was evaluated primarily on latency. RAGCache addresses that gap directly by caching intermediate knowledge representations, reporting a reduction in time-to-first-token of up to 4x and a throughput improvement of up to 2.1x relative to a vLLM-plus-Faiss baseline [16]. Prompt Cache generalizes the idea to reusable attention states across prompts sharing a common structure, reporting time-to-first-token improvements ranging from 8x on GPU-based inference to 60x on CPU-based inference [17].

What this literature shares, more than any specific technique, is scope. The unit of measurement is almost always the model call: tokens per second, time to first token from a warmed cache, or throughput at a fixed batch size. That scope suits research aimed at model-serving infrastructure, and the gains are real, several of them substantial. It becomes a limitation only when the results are extrapolated to end-to-end enterprise latency, because a production conversational turn includes stages these benchmarks do not model.

We express that end-to-end latency with a pipeline decomposition formula developed from operating large-scale conversational AI platforms: $TM99 = \text{WebSocket} + \max(\text{LLM Routing}, \text{Contextualization}) + \max(\text{Cache Lookup}, \text{Guardrail}) + \text{Plugin Execution} + \text{Response Synthesis}$. The formula separates stages that must execute sequentially from stages that can run concurrently. WebSocket connection establishment happens once, at the start of the turn, and cannot overlap with anything downstream. Plugin execution and response synthesis happen at the end and depend on the outputs of everything before them, so they too remain sequential. The routing decision, by contrast, does not need to wait for contextualization, and cache lookup does not need to wait for the guardrail check; both pairs can execute in parallel, with the slower branch in each pair setting the pace. The nested max terms capture this: total latency is governed by the longest path through the pipeline, not the sum of every stage's individual latency. A system that optimizes each stage in isolation, without exploiting this parallelism, will report improvement in every component benchmark while barely moving the metric users actually experience.

Table 1 summarizes how each stage in this formula behaves, which stages can run concurrently, and where the corresponding optimization lever sits.

Table 1. Pipeline stage latency characteristics in the TM99 decomposition formula.

Pipeline Stage	Execution Position	Parallelizable With	Primary Latency Driver	Optimization Lever
WebSocket Handshake	Sequential (start)	None	Network round-trip, TLS negotiation	Connection pooling, persistent sessions
LLM Routing Decision	Parallel branch	Contextualization	Classifier inference, tier selection logic	Lightweight routing models, learned routers
Contextualization (Retrieval)	Parallel branch	LLM Routing	Vector search, document fetch	Index sharding, approximate nearest-neighbor search
Cache Lookup	Nested parallel	Guardrail Check	Cache key matching, semantic similarity search	Semantic caching, prompt/attention-state caching

Pipeline Stage	Execution Position	Parallelizable With	Primary Latency Driver	Optimization Lever
Guardrail / Safety Check	Nested parallel	Cache Lookup	Policy model inference, rule evaluation	Lightweight classifiers, asynchronous post-hoc screening
Plugin Execution	Sequential (post-convergence)	None	External API latency, third-party dependency	Timeout budgets, speculative pre-fetch
Response Synthesis / Streaming	Sequential (end)	None	Token generation, output formatting	Streaming transport, speculative decoding

3. Three-Path Execution Model and RAG Latency Sub-Components

The parallelism implied by the pipeline decomposition formula is not automatic. It has to be built into the request-handling architecture, and in production systems this typically takes the shape of three concurrent execution paths that begin the moment a user message arrives and reconverge before response synthesis.

The first path is the routing path. A routing layer classifies the incoming request, decides which model tier should handle it, and determines whether the request needs tool access, retrieval, or can be answered directly. This decision can begin as soon as the message is parsed, and it does not depend on the outcome of retrieval or safety screening. The second path is the contextualization path, which retrieves relevant documents, conversation history, and user or session state needed to ground the response. The third path is the guardrail path, which screens the incoming request and, later, the generated response for policy violations, prompt injection attempts, and compliance risks. Within the contextualization path itself, cache lookup and the guardrail check can run concurrently, since a semantic cache hit and a safety violation are independent conditions, neither of which needs to wait on the other to be evaluated.

Running these paths concurrently rather than sequentially makes the nested max structure in the pipeline formula achievable in practice rather than theoretical. When routing, contextualization, and guardrail evaluation are implemented as sequential middleware, as they commonly are in early-stage conversational AI systems, their latencies simply add together, and the resulting TM99 can run two to

three times higher than the same system with concurrent execution, holding every individual component constant. In the experience underlying this paper, this is the single largest architectural lever available to teams inheriting a conversational AI stack that grew incrementally rather than being designed for concurrency from the outset.

Streaming infrastructure changes what latency means for the end user and, correspondingly, what should be measured. A non-streaming request has a single meaningful timestamp: the moment the complete response returns. The minimum number of latencies that occur during a stream is two: first-byte latency when the first token or chunk becomes available, and last-byte latency when the last token becomes available. These two numbers behave differently and respond to different optimization levers. First-byte latency is dominated by everything upstream of generation: connection setup, routing, retrieval, cache lookup, and guardrail evaluation on the input side. Last-byte latency adds the full generation time, streamed token by token, plus any output-side guardrail checks applied to the completed or partially completed response.

For a user, first-byte latency governs the perception of responsiveness. A response that begins streaming within a few hundred milliseconds feels immediate even if the full answer takes several seconds to complete, because the user's attention shifts from waiting to reading. Last-byte latency governs task completion time and matters most for workflows where the user or a downstream system needs the complete answer before proceeding, such as when a plugin call depends on the full generated output rather than a partial stream. A platform that reports only end-to-end latency conflates these two

experiences. It can miss the fact that a system with excellent first-byte performance and mediocre last-byte performance may still satisfy most interactive use cases, while a system with the reverse profile feels sluggish even when its total latency number looks comparable.

Retrieval-augmented generation adds its own sub-components to this picture, each with a distinct latency signature. Query embedding converts the user's message into a vector representation and typically completes in low single-digit milliseconds on modern hardware. Vector search against a knowledge index follows, with latency that scales with index size, the number of shards queried, and the similarity search algorithm in use. Document reranking, when present, adds a further pass over the initial candidate set to improve relevance before the documents enter the model's context window. Each of these sub-components can be cached, batched, or parallelized independently. RAGCache's caching of intermediate knowledge representations reduces time-to-first-token by up to 4x and improves throughput by up to 2.1x compared with a vLLM-plus-Faiss baseline [16], and Prompt Cache's reuse of attention states across structurally similar prompts produces time-to-first-token gains ranging from 8x on GPU-based systems to 60x on CPU-based systems [17]. Both results depend on cache hit rates that vary substantially with query diversity, a point the discussion returns to below.

The three-path model, combined with an explicit first-byte and last-byte distinction, gives engineering teams a vocabulary for locating where latency is actually spent, rather than treating the request as an undifferentiated interval between user input and final output. That vocabulary is a prerequisite for the benchmarking and optimization discussion that follows.

4. Comparative Analysis and Discussion

Three categories of technique account for most of the measured latency and cost gains reported in the reference literature: caching, tiered or speculative routing, and phase-level serving optimizations. Each behaves differently once moved from a benchmark environment into a production enterprise pipeline carrying multiple concurrent agents, tool calls, and compliance requirements. Table 2 collects the reported gains for the techniques discussed in this section alongside their sources.

Caching delivers the largest reported multiples, and also the widest variance. GPTCache's semantic caching increases response speed by 2 to 10 times when a cache hit occurs [8], and Prompt Cache reports time-to-first-token improvements ranging from 8x on GPU-based inference to 60x on CPU-based inference [17]. RAGCache, targeting the retrieval side specifically, reduces time-to-first-token by up to 4x and improves throughput by up to 2.1x against a vLLM-plus-Faiss baseline [16]. These figures are accurate as reported, and semantic caching is genuinely one of the highest-leverage optimizations available to a conversational AI platform. The qualifier that matters in production sits in the phrase "when a cache hit occurs." Enterprise conversational traffic, particularly in multi-agent settings where each agent handles a narrower and more specialized query distribution, tends to show lower cache hit rates than the broad consumer-facing traffic these benchmarks typically evaluate against. A support bot answering a small set of frequently asked questions can reach cache hit rates that make a 60x improvement meaningful across a large share of total traffic. An internal engineering assistant answering open-ended, context-specific questions about a unique codebase will hit cache far less often, and the same technique that transforms one workload barely moves another.

Model tiering and speculative routing show a more consistent, if more modest, transfer to enterprise settings. RouteLLM reduces costs by over 2 times in certain cases without compromising quality by learning to route queries between a strong and a weak model [18], and HybridLLM's query router achieves up to 40% fewer calls to the large model with no drop in response quality [19]. FrugalGPT extends the cascade idea further, matching the performance of the best individual large language model with up to 98% cost reduction, or improving accuracy over GPT-4 by 4% at the same cost, depending on the operating point selected [20]. These results are primarily cost-optimization findings, and cost and latency are related but not identical; routing to a smaller model reduces both compute cost and, generally, generation latency, but the routing decision itself adds a small fixed latency cost that these papers do not always report against a strict TM99 target. Running routing layers at production scale suggests that this fixed routing cost is real but small relative to the downstream savings, provided the routing decision executes in parallel with contextualization rather than blocking it, which

is exactly the parallelism the pipeline decomposition formula is built to exploit.

Any discussion of speed gains in retrieval-augmented settings needs an accuracy check attached to it. Chen et al.'s benchmarking study of large language models in retrieval-augmented configurations finds substantial variation in how retrieval and generation choices affect both accuracy and response time [21]. The CRAG benchmark sharpens the concern further: the most advanced language models achieve no more than 34% accuracy on its comprehensive question set, adding retrieval-augmented generation improves accuracy only to 44%, and even state-of-the-art industry RAG solutions answer only 63% of questions without hallucination [22]. These figures matter here because they establish a ceiling on how aggressively retrieval and generation can be short-circuited for speed. A system tuned purely for first-byte latency, at the expense of retrieval thoroughness, risks trading a real latency gain for a real accuracy loss, and the CRAG results suggest that margin is already thin before any speed-motivated shortcuts are applied.

Speculative decoding and speculative sampling sit closer to the model-serving layer and transfer well because they change how tokens are generated without changing anything about the surrounding orchestration. Speculative decoding reports a 2x-3x wall-clock speedup on T5-XXL with no architectural changes and no retraining [6], and speculative sampling reports a 2-2.5x decoding speedup on Chinchilla 70B [7]. Because these techniques operate entirely within the model-serving boundary, their gains hold regardless of how many agents, plugins, or guardrail checks surround the model call, which makes them among the most portable optimizations in the reference set.

This is where the paper's central argument has to confront its strongest counterpoint directly. Model-level optimization techniques, including speculative decoding, quantization-adjacent serving improvements, and paged memory management, are more general-purpose than orchestration-layer changes: they apply to essentially any deployment using the underlying model, independent of how a given organization has structured its conversational pipeline. Orca's 36.9x throughput improvement at the same level of latency [4] and PagedAttention's 2-4x throughput gain at matched latency [5] benefit every request flowing through the serving layer,

with no architectural assumptions about routing, caching, or guardrail placement. Orchestration-layer parallelism, by contrast, has to be re-implemented for each organization's specific pipeline topology, and the size of the gain depends entirely on how sequential that pipeline was to begin with. A team that has already parallelized routing and contextualization has little left to gain from further orchestration work and should turn to model-serving techniques next.

The counterargument holds as far as it goes, and it explains why model-level research dominates the published literature: it produces results that generalize across deployments, which is what publication and benchmarking incentives reward. It does not, however, describe the state of most enterprise conversational AI platforms observed in production. The large majority of these systems, particularly those grown through incremental feature addition rather than a unified latency-first design, still execute routing, contextualization, and guardrail checks sequentially, and the resulting TM99 gap between sequential and parallel execution of these three stages routinely exceeds the gains available from any single model-serving technique in this reference set. In that common condition, orchestration-layer parallelism is the higher-leverage investment. Not because model-level techniques are less effective in principle, but because most production pipelines have not yet captured the orchestration-layer gains available to them, while model-serving improvements have already been captured on the platform's behalf by upstream vendors and open-source serving frameworks such as vLLM [5] and DistServe [13].

Measuring these gains correctly requires abandoning mean latency as the primary metric. A mean blends fast and slow requests into a single number that no individual user experiences, and it is disproportionately sensitive to a small number of very fast cache-hit responses that can mask a long tail of slow, cache-miss, multi-agent requests. Table 3 lays out how the three benchmarking percentiles used throughout this paper differ in what they represent and where they point optimization effort. TM50, the median, describes the typical user's experience and suits general user-experience tuning. TM90 captures the experience of a meaningfully large minority operating under moderate load or query complexity, and is where cache-miss and moderate-complexity routing paths start to become

visible. TM99 captures the worst-case tail, the requests service-level objectives are actually written against, and is where sequential-versus-parallel pipeline architecture, guardrail evaluation cost, and plugin timeout behavior dominate the result. A platform that reports an improved mean latency while its TM99 stays flat, or worsens, has very likely

improved the common case at the expense of the cases that drive support escalations, SLO violations, and user attrition. That is precisely the failure mode percentile-based benchmarking is built to catch and mean-based benchmarking is structurally unable to detect.

Table 2. Comparison of caching, tiering, and serving techniques and their measured gains.

Technique	Representative System	Reported Gain	Source
Semantic caching	GPTCache	2-10x response speed increase on cache hit	[8]
Prompt / attention-state caching	Prompt Cache	8x (GPU) to 60x (CPU) time-to-first-token reduction	[17]
RAG-specific caching	RAGCache	Up to 4x TTFT reduction; 2.1x throughput improvement	[16]
Speculative decoding	Leviathan et al.	2-3x wall-clock speedup (T5-XXL)	[6]
Speculative sampling	Chen et al.	2-2.5x decoding speedup (Chinchilla 70B)	[7]
Learned model routing	RouteLLM	Over 2x cost reduction without compromising quality	[18]
Query-complexity routing	HybridLLM	Up to 40% fewer large-model calls, no quality drop	[19]
Cascade routing	FrugalGPT	Up to 98% cost reduction at matched performance	[20]
Multi-level inference	Tabi	21%-40% reduction in average latency	[11]
Paged memory management	vLLM / PagedAttention	2-4x throughput at the same level of latency	[5]
Continuous batching	Orca	36.9x throughput improvement at the same level of latency	[4]
Prefill/decode disaggregation	DistServe	7.4x more requests or 12.6x tighter SLO	[13]

Table 3. TM50/TM90/TM99 optimization targets and their diagnostic purposes.

Metric	Represents	Primary Diagnostic Use	Optimization Focus
TM50 (median)	Typical user experience	General UX tuning, baseline health check	Common-path efficiency
TM90	Majority-tail experience	Moderate load and cache-miss behavior	Routing efficiency, cache-miss handling
TM99	Worst-case tail	SLO compliance, escalation-driving requests	Orchestration parallelism, guardrail cost, plugin timeouts
Mean latency	Blended average across all requests	Not recommended as a primary SLO metric	Masks tail behavior; skewed by cache hits

5. Conclusion

Three conclusions frame where enterprise conversational AI teams should direct optimization effort next. First, the pipeline decomposition formula presented here shows that end-to-end latency, particularly at the TM99 tail, is governed by the longest path through a set of stages that include but are not limited to model inference. WebSocket setup, routing, contextualization, guardrail evaluation, plugin execution, and response synthesis all contribute, and several of them can run concurrently rather than sequentially. Second, the measured gains reported across the caching, routing, and speculative-execution literature reviewed here are real: GPTRCache's semantic cache increases response speed 2-10x on a hit [8], Prompt Cache cuts time-to-first-token 8x to 60x depending on hardware [17], and DistServe serves 7.4x more requests within latency constraints through prefill-decode disaggregation [13]. Their transfer to enterprise multi-agent settings, however, depends heavily on cache hit rates and pipeline topology that vary widely across organizations. Third, TM99-based benchmarking, not mean latency, is the appropriate lens for evaluating any of these techniques, because the tail is where architectural bottlenecks and service-level violations actually live.

The practical implication is a shift in emphasis, not a rejection of model-level work. Quantization, distillation, and efficient serving remain valuable, portable, and worth adopting regardless of an organization's orchestration maturity. For the many enterprise platforms still running routing, contextualization, and safety checks in sequence, though, the orchestration and routing layer is where the next order-of-magnitude improvement in TM99 is most likely to be found, and that improvement is available without waiting on the next generation of model-serving research.

Teams building or operating large-scale conversational AI platforms would do well to instrument first-byte and last-byte latency separately, benchmark at TM50, TM90, and TM99 rather than at the mean, and audit their own pipelines for sequential stages that could be re-architected as concurrent paths before assuming the next latency win has to come from the model itself. As conversational AI systems absorb more concurrent users and more complex multi-agent workflows, that orchestration-layer discipline will matter as much as any advance in the underlying models.

Ethics Statement

The author declares no conflict of interest. This study did not involve human participants, animal subjects, or identifiable personal data requiring

ethics board review; informed consent was not applicable to this work. The research draws entirely on published, publicly available benchmarking literature and the author's own architectural and operational experience with production conversational AI infrastructure.

Acknowledgments

The author received no external funding or institutional support for this work.

Author Contributions

Prasun Bhattacharyya is the sole author and is responsible for all aspects of the conceptualization, analysis, writing, and review of this manuscript.

Conflicts of Interest

The author declares no conflict of interest.

References

- [1] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 5998-6008.
- [2] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459-9474.
- [3] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," *Proceedings of Machine Learning and Systems*, vol. 5, 2023.
- [4] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for transformer-based generative models," in *Proc. 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022, pp. 521-538.
- [5] W. Kwon et al., "Efficient memory management for large language model serving with PagedAttention," in *Proc. 29th ACM Symposium on Operating Systems Principles (SOSP)*, 2023.
- [6] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *Proc. 40th International Conference on Machine Learning (ICML)*, 2023.
- [7] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper, "Accelerating large language model decoding with speculative sampling," *arXiv preprint arXiv:2302.01318*, 2023.
- [8] F. Bang, "GPTCache: An open-source semantic cache for LLM applications," in *Proc. 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS)*, 2023, pp. 212-218.
- [9] V. J. Reddi et al., "MLPerf inference benchmark," in *Proc. ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 446-459.
- [10] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, H. Jin, T. Chen, and Z. Jia, "Towards efficient generative large language model serving: A survey from algorithms to systems," *arXiv preprint arXiv:2312.15234*, 2023.
- [11] Y. Wang, K. Chen, H. Tan, and K. Guo, "Tabi: An efficient multi-level inference system for large language models," in *Proc. 18th European Conference on Computer Systems (EuroSys)*, 2023, pp. 233-248.
- [12] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," in *Proc. International Conference on Learning Representations (ICLR)*, 2024.
- [13] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, "DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *Proc. 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.
- [14] P. Patel, E. Choukse, C. Zhang, A. Shah, I. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative LLM inference using phase splitting," in *Proc. ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 118-132.
- [15] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-RAG: Learning to retrieve, generate, and critique through self-reflection," in *Proc. International Conference on Learning Representations (ICLR)*, 2024.
- [16] C. Jin, Z. Zhang, X. Jiang, F. Liu, X. Liu, X. Liu, and X. Jin, "RAGCache: Efficient knowledge caching for retrieval-augmented generation," *arXiv preprint arXiv:2404.12457*, 2024.
- [17] I. Gim, G. Chen, S. Lee, N. Sarda, A. Khandelwal, and L. Zhong, "Prompt cache: Modular attention reuse for low-latency inference," *Proceedings of Machine Learning and Systems*, vol. 6, 2024.

- [18] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica, "RouteLLM: Learning to route LLMs with preference data," in Proc. International Conference on Learning Representations (ICLR), 2025.
- [19] D. Ding, A. Mallick, C. Wang, R. Sim, S. Mukherjee, V. Ruhle, L. V. S. Lakshmanan, and A. H. Awadallah, "Hybrid LLM: Cost-efficient and quality-aware query routing," in Proc. International Conference on Learning Representations (ICLR), 2024.
- [20] L. Chen, M. Zaharia, and J. Zou, "FrugalGPT: How to use large language models while reducing cost and improving performance," arXiv preprint arXiv:2305.05176, 2023.
- [21] J. Chen, H. Lin, X. Han, and L. Sun, "Benchmarking large language models in retrieval-augmented generation," in Proc. AAAI Conference on Artificial Intelligence, vol. 38, no. 16, 2024.
- [22] X. Yang et al., "CRAG: Comprehensive RAG benchmark," in Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track, 2024.