# CAGE: An Evaluation Framework for Cache-Augmented Generation Models

**Lucas Mariano do Carmo, Wladmir Cardoso Brandão, Henrique Cota de Freitas**

Graduate Program in Informatics, Department of Computer Science
Pontifícia Universidade Católica de Minas Gerais (PUC Minas), Belo Horizonte, Brazil

lucas.mariano@sga.pucminas.br, wladmir@pucminas.br, cota@pucminas.br

***Abstract.*** *Cache-Augmented Generation (CAG) is an emerging design that reduces the cost of repeated prompt processing by reusing previously processed context, yet it lacks a standard evaluation approach. We present CAGE, a framework that combines serving metrics and semantic quality analysis across baselines in cache-aware AI systems. CAGE integrates features from vLLM's native prefix caching and evaluates latency, TTFT, throughput, and semantic metrics. In our results, native prefix caching reduced latency of 37.4% and TTFT by 65.7% with no loss in faithfulness, whereas RAG increased latency by 70.4% and reduced faithfulness by 11.6%. These results validate the usefulness of CAGE as an approach for evaluating cache-aware LLM systems.*

## 1. Introduction

Large Language Models (LLMs) are used widely nowadays in systems that must answer questions, explain technical outputs, summarize documents, and generate code. In these settings, answer quality depends not only on the model itself, but also on how context is supplied during inference. Retrieval-Augmented Generation (RAG) has become the standard approach for grounding responses in external knowledge [Souza et al. 2025], but it adds latency, retrieval error, and pipeline complexity [Lewis et al. 2020].

However, the design space is shifting from pure single-turn retrieval to dynamic, context-persistent, multi-turn reasoning that reuses stable context across requests. In this work, we use Cache-Augmented Generation (CAG) to refer to approaches that improve inference efficiency by reusing previously processed context or cached model states when relevant context is stable or repeatedly used, while treating retrieval-backed hybrids as a separate comparison family [Chan et al. 2025; Lu et al. 2025].

Despite growing interest in CAG, evaluation practice and research have been primarily on the RAG literature. Frameworks such as RAGAS, ARES, BERGEN, and RAGBench are useful for retrieval-based systems, but they were not designed to measure locality, cache hit ratios, time-to-first-token (TTFT) behavior, or the quality implications of reusing internal context representations

The authors would like to thank the Brazilian National Council for Scientific and Technological Development (CNPq; grant numbers 311697/2022-4 and 402837/2024-0), the Coordination for the Improvement of Higher Education Personnel - Brazil (CAPES; grant numbers PROAP 88887.842889/2023-00 - PUC/MG, PDPG 88887.708960/2022-00 - PUC/MG - Informatics, and Finance Code 001), the Research Support Foundation of the State of Minas Gerais (FAPEMIG; grant number APQ-05058-23), and PUC Minas (FIP grant number 2026/35258).

[Es et al. 2024; Friel et al. 2024; Saad-Falcon et al. 2024; Rau et al. 2024]. Based on this issue, a key question arises: how should cache-aware systems be evaluated when latency, reuse behavior, and answer quality must be analyzed jointly?

The goal of this paper is to propose and analyze a modular CAG Evaluation (CAGE) framework that combines cache and retrieval telemetry with semantic quality analysis in a single approach. The main contributions are the systematic comparison of cache-backed, retrieval-backed, hybrid, and distributed baselines under common conditions, and the identification of trade-offs among reuse efficiency, serving latency, and answer quality, which together constitute the CAGE features.

This work is organized as follows: Section 2 discusses the Key-Value cache, the foundational concept behind cache-aware inference, and reviews related work. Section 3 presents the framework architecture, baseline definitions, and experimental setup. Section 4 reports the experimental results across serving performance, tail latency, quality, and cache telemetry. Section 5 discusses the overall findings, and we conclude by presenting our conclusions and directions for future work.

## 2. Understanding Key-Value Cache as the System-Level Core

The Key-Value (KV) Cache is the central mechanism that enables autoregressive generation with Transformer models for long sequences [Kwon et al. 2023]. It is the technological backbone of the CAG paradigm, transforming a compute-bound problem into one dominated by memory bandwidth.

While solving a computational problem, the KV Cache introduces a massive memory problem; its size grows linearly with the sequence length and batch size, often consuming more GPU VRAM than the model weights themselves. For a large model like Llama 3.1 70B with a 128K context window, the cache for a single sequence can reach 40 GB. The recognition of the KV Cache as a primary bottleneck has spurred significant research [H. Li et al. 2024]. These strategies can be categorized into single-node and distributed techniques.

Single-node KV cache optimization focuses on reducing the memory pressure caused by long-context inference within a single accelerator or machine. These techniques do not distribute the cache across devices; instead, they improve how memory is represented, allocated, and retained locally [Javidnia et al. 2025]. Distributed KV cache strategies leverage multiple GPUs or nodes to store and process a much larger shared cache. Rather than only compressing or managing the cache locally, these approaches distribute its storage and associated attention computation across devices. [Qin et al. 2024].

Understanding these KV cache strategies is essential to CAGE's design, as the framework must measure how each strategy affects not only latency and throughput, but also the quality of the generated answers. The present validation covers single-node prefix caching and retrieval-backed baselines, establishing the metrics and architecture, while subsequent phases will extend to distributed KV transfer and disaggregated serving, where trade-offs between memory management, transfer cost, and answer quality become significantly more complex.

### 2.1. Related Work

A fundamental limitation of Large Language Models (LLMs) is their reliance on static, parametric knowledge acquired during their extensive pre-training phase. This knowledge is inherently frozen in time and lacks domain specificity, leading to challenges such as inaccuracies ("hallucinations") and an inability to reason over proprietary or real-time data [Sahoo et al. 2024; Barnett et al. 2024].

Cache-aware inference changes the unit of optimization from raw model execution to memory reuse. vLLM [Kwon et al. 2023] uses memory management techniques like PagedAttention to reduce fragmentation and support efficient prefix reuse. This systems view is the minimal background needed for CAGE, because it explains why metrics such as TTFT, latency, throughput, and reuse rate matter alongside answer quality. From an evaluation perspective, this changes the nature of the bottleneck, as generation becomes less about raw model execution in isolation and more about how memory and context are managed across requests.

RAGAs [Es et al. 2024] is a reference-free evaluation framework for retrieval-augmented pipelines. Its contribution is a metric design for context relevance, faithfulness, and answer relevance, but it does not measure cache locality, reuse ratio, or the serving-time effects of cache-aware execution.

RAGBench [Friel et al. 2024] is a benchmark for retrieval-grounded systems that formalizes the TRACe evaluation framework. It is highly relevant for benchmarking, but it remains focused on retriever-generator behavior rather than on the trade-offs that arise when knowledge is reused from cache rather than retrieved online.

Chan et al. [Chan et al. 2025] argued that cache-augmented generation can replace or complement retrieval when the knowledge base is limited and stable enough to be loaded in advance. Their results motivate CAG as a serious alternative to retrieval-heavy pipelines, but the paper does not define a general approach for measuring latency, reuse behavior, and semantic quality across multiple cache-aware baselines

TurboRAG [Lu et al. 2025] moved closer to the systems side by precomputing document KV states offline and reusing them during retrieval-time inference. This work shows that retrieval and cache reuse can be combined, yet its contribution is a serving method rather than an evaluation framework for comparing cache policies and semantic outcomes under common workloads.

DistServe study [Zhong et al. 2024] disaggregated prefill and decode serving, treating TTFT and per-token latency as separate systems objectives and assigning the two phases to different GPUs. Their work is foundational for the multi-node setting targeted by CAGE, but it optimizes serving goodput rather than jointly evaluating serving behavior and answer quality.

Self-Route [Z. Li et al. 2024] compared retrieval-based and long-context processing to propose how to choose between them based on model self-reflection. This work is also relevant to CAGE's hybrid evaluation direction, and it frames routing as a cost-quality trade-off, but it does not measure cache telemetry, locality, or distributed reuse effects.

As shown in Table 1, related work typically targets one axis at a time. CAGE

(Figure 1) differs in that it jointly evaluates performance, cache telemetry, and semantic quality across cache-backed and retrieval-backed baselines using a common approach.

**Table 1. Comparison with adjacent work.**

| Work | Semantic quality | Systems metrics | Cache reuse | Distributed |
|---|---|---|---|---|
| RAGAS [Es et al. 2024] | Yes | No | No | No |
| RAGBench [Friel et al. 2024] | Yes | No | No | No |
| TurboRAG [Lu et al. 2025] | Partial | Yes | Yes | No |
| DistServe [Zhong et al. 2024] | No | Yes | No | Yes |
| SelfRoute [Z. Li et al. 2024] | Partial | Limited | No | No |
| This Work | Yes | Yes | Yes | Yes |

## 3. CAGE Framework Architecture - System Design

The framework was designed as a layered testbed (Figure 1) with a workload layer that generates requests from a controlled dataset split, and an orchestration layer that selects the baseline, manages the run configuration, and enforces consistent experimental conditions across trials. A telemetry layer records system behavior during execution, including latency, TTFT, TPOT, throughput, and cache or retrieval signals. The quality layer scores generated answers against references and supporting context using metrics such as faithfulness, relevance, BERTScore, and ROUGE-L. Finally, the analysis layer aggregates trial outputs into summary tables and plots.

This separation is important because it prevents system and model or dataset changes, it also keeps the framework modular, so that new components or pipelines can be introduced without changing the evaluation logic itself. Redis [Redis 2025] is used in CAGE as an optional centralized retrieval-cache baseline and in hybrid retrieval experiments, not as the primary KV-cache mechanism of the architecture.

### 3.1. Baseline Setting

Our initial validation used seven baselines that vary in both the context source and the reuse policy.

*No Cache* uses the gold SQuAD passage and recomputes every request from scratch. One clarification: *No Cache* does not mean *No Content*; it means *No Reuse*. It serves as the gold-context recomputation control, isolating the recomputation cost while retaining the best available evidence in the prompt.

*Prefix Cache* uses the same gold passage but enables native vLLM prefix reuse. It is the simplest single-node caching baseline and measures the benefit of reuse without external cache infrastructure.

*RAG* replaces the gold passage path with retrieved passages obtained through dense retrieval and reranking. It represents the standard retrieval-based design and serves as the main comparison point for systems that rely on external search rather than cache reuse.

*Redis Retrieval Cache Cold* relies on retrieved artifacts and stores them through a centralized Redis layer. The Redis cache is flushed before each run to ensure cold-start

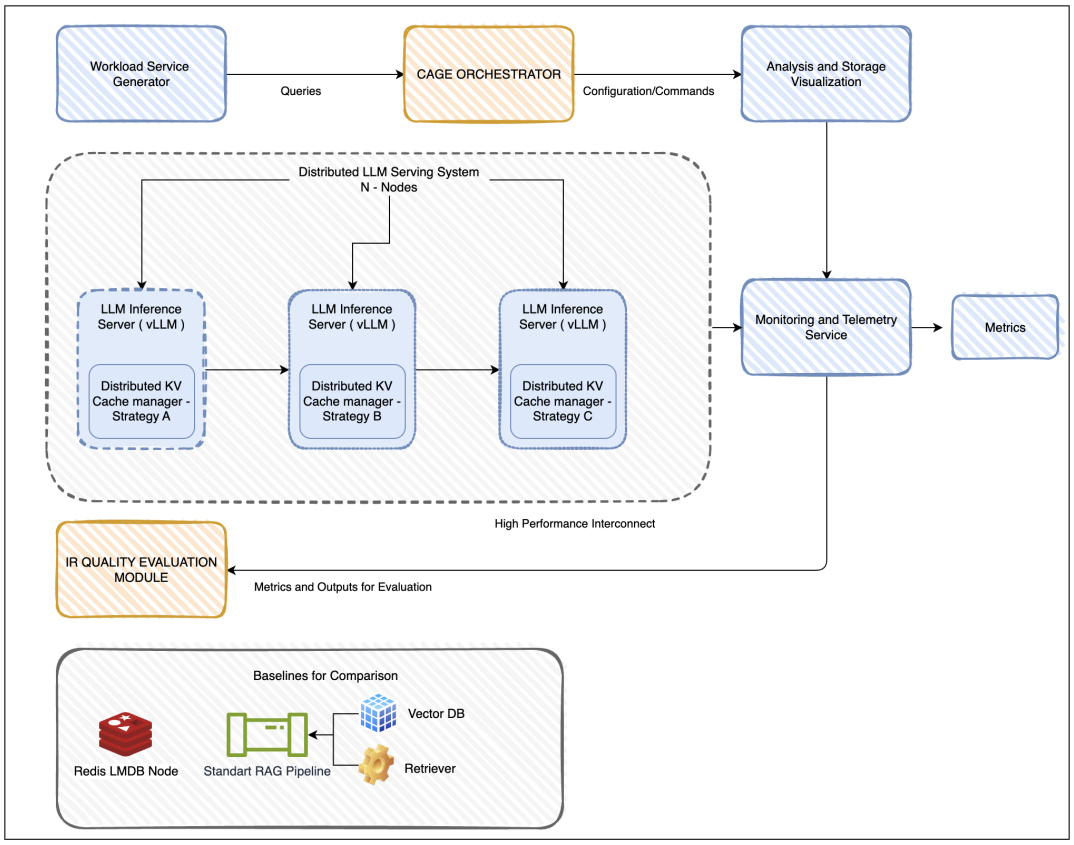


**Figure 1. The CAGE Framework Architecture Diagram, illustrated step-by-step.**

semantics. This baseline measures centralized retrieval-artifact reuse with external lookup overhead. Importantly, it does not store raw vLLM KV tensors in Redis; it evaluates centralized reuse within the retrieval layer.

*Hybrid Retrieval Cache Cold* combines retrieval with native prefix caching. The retrieval cache starts cold (flushed), but the vLLM prefix cache remains active. This baseline isolates the effect of prompt-prefix reuse on top of retrieval.

*Hybrid Retrieval Cache Warm* is the same hybrid design but with an explicit warmup phase. Fifty warmup queries are executed and excluded from measured metrics, ensuring both the retrieval cache and prompt cache are warm during evaluation.

*Distributed Router* returns to the gold passage and evaluates a multi-replica routing design with real tokenized prefix hashing. The role is to demonstrate orchestration-level distributed routing rather than cross-node KV tensor transfer.

### 3.2. Local Setup and Initial configuration

Our use of SQuADv2 is backed by Chan et al. [Chan et al. 2025], and Qwen/Qwen3-4B as a model that is feasible for controlled CPU-side evaluation. Therefore, initial validation can be based on a benchmark that is sufficiently standardized for comparison and sufficiently controlled for baseline analysis. This choice was designed strictly as a controlled validation step to verify the integrity of the CAGE orchestration logic, specifically, and to determine whether our telemetry correctly captures the correlation between prompt-cache hit ratios and Time-To-First-Token (TTFT) reductions. Because this local setup does not

induce the extreme KV-cache memory pressure that motivates Cache-Augmented Generation (CAG), we explicitly state that the absolute latency values are not generalizable.

Each baseline was evaluated with 50 measured queries per trial across three sequential trials, yielding 1050 measured queries in total (7 baselines $\times$ 50 queries $\times$ 3 trials). For prefix-cache-enabled baselines, the server was restarted per baseline rather than per trial, so later trials may inherit warmed prompt-cache state and should not be interpreted as fully independent cold starts. The warm hybrid baseline also performs additional warmup queries, which are excluded from the reported metrics.

Inference was performed on the vLLM CPU backend (Apple M4 Pro, 24 GB unified memory, 12 cores) with a maximum of 100 output tokens and non-greedy decoding (temperature 0.7, top-$p$ 0.95). Retrieval used intfloat/e5-large-v2 for embedding, BAAI/bge-reranker-large for reranking, and top-$k$ = 3. The KV cache was allocated 10 GB via an environment variable, sufficient to exercise reuse behavior within the available memory. The CPU-only setup is an initial constraint that keeps system overhead visible and reproducible; GPU infrastructure will be used to validate absolute latency values and distributed effects at scale in a later execution setting.

We intentionally included BERTScore not as a primary metric, but as a negative empirical control. It serves as a naive baseline for semantic and structural similarity, helping us demonstrate why advanced evaluation frameworks require strict, NLI-based entailment (Faithfulness) to catch hallucinations that soft-matching metrics miss. This methodological choice is supported by recent literature. The RAGAs framework [Es et al. 2024] was developed specifically because the authors identified that metrics like BERTScore fail to assess factual hallucinations, prompting a shift toward LLM-as-a-judge methodologies to capture true "Faithfulness." Similarly, the ARES framework [Saad-Falcon et al. 2024] utilizes fine-tuned NLI judges, because standard embedding similarity cannot differentiate between a passage that is merely topically similar and one that factually answers the prompt.

## 4. Experimental Results

This section reports the initial local validation of CAGE, designed to establish baseline results and verify the framework architecture. All seven baselines were evaluated on a single CPU node under the conditions described in Section 3.2.

### 4.1. Serving Performance

Table 2 summarizes the serving metrics across all seven baselines. Prefix Cache was the best overall baseline, achieving 37.4% lower latency (10,015ms versus 16,006ms), 65.7% lower TTFT (2376ms versus 6919ms), and 58.3% higher throughput (0.096 QPS versus 0.061 QPS). This is due to the native prefix reuse in vLLM, which avoids redundant prompt processing without altering the context source.

**Table 2. Performance Metrics (Mean ± Std Dev, n=50 queries, 3 trials)**

| Baseline | QPS | Latency (ms) | TTFT (ms) | TPOT (ms) | Tok/s |
|---|---|---|---|---|---|
| No Cache | 0.061 ± 0.003 | 16,006 ± 711 | 6919 ± 112 | 90.9 ± 6.5 | 6.1 |
| Prefix Cache | **0.096 ± 0.008** | **10,015 ± 884** | **2376 ± 275** | **76.4 ± 9.6** | **9.6** |
| RAG | 0.033 ± 0.001 | 27,270 ± 283 | 18,775 ± 144 | 84.9 ± 1.5 | 3.3 |
| Redis Cache Cold | 0.034 ± 0.000 | 26,853 ± 39 | 18,579 ± 37 | 82.7 ± 0.1 | 3.4 |
| Hybrid Cache Cold | 0.059 ± 0.012 | 15,513 ± 4173 | 6001 ± 4290 | 95.1 ± 2.0 | 5.9 |
| Hybrid Cache Warm | 0.067 ± 0.004 | 13,269 ± 893 | 2791 ± 66 | 104.8 ± 8.4 | 6.7 |
| Distributed Router | 0.052 ± 0.001 | 18,492 ± 372 | 5279 ± 638 | 132.1 ± 3.5 | 5.2 |

The Distributed Router baseline improved TTFT by 23.7% relative to No Cache (5279ms versus 6919ms), showing that multi-replica routing with prefix-aware hashing can improve responsiveness, although its total latency remains 15.5% higher than No Cache due to routing overhead.

RAG incurred 70.4% higher latency than No Cache, driven by retrieval and reranking overhead. The hybrid baselines show that combining retrieval with prefix caching can recover part of this cost: Hybrid Cache Warm achieved 17.1% lower latency than No Cache, though it still cannot match the pure Prefix Cache baseline.

The TPOT column reveals that decode speed does not follow the same ranking as prefill. Prefix Cache achieved the lowest TPOT (76.4 ms/token) and the highest token throughput (9.6 tok/s), while Distributed Router has the highest TPOT (132.1 ms/token) despite its TTFT advantage. Among retrieval baselines, RAG and Redis Cache Cold show moderate TPOT (84.9 and 82.7 ms), but their total latency is dominated by the prefill phase rather than by decode cost.

Figures 2 and 3 visualize the serving-performance ranking from Table 2.

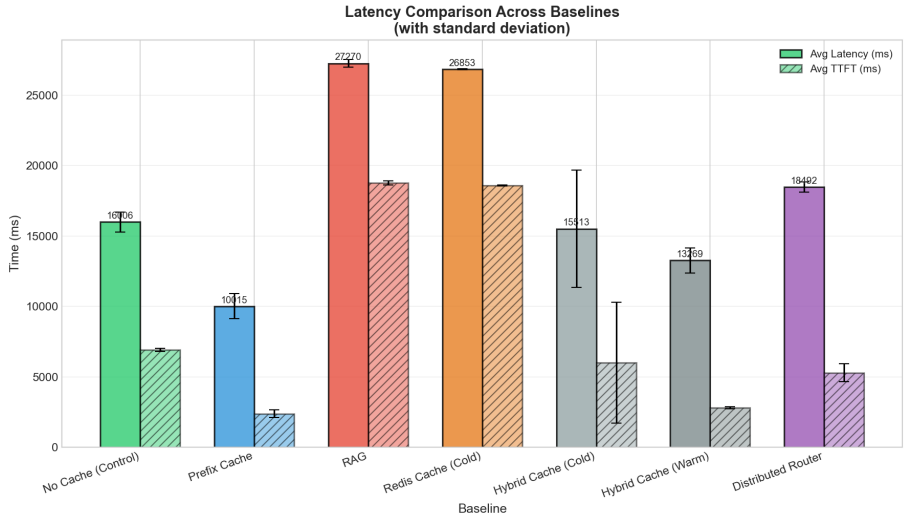


**Figure 2. Average latency and TTFT across baselines (with standard deviation).**

Figure 2 makes visible the separation between cache-aware and retrieval-dominated baselines: the two retrieval settings are nearly $2.7\times$ slower than the control, whereas Prefix Cache reduces latency to $0.63\times$ of the control.

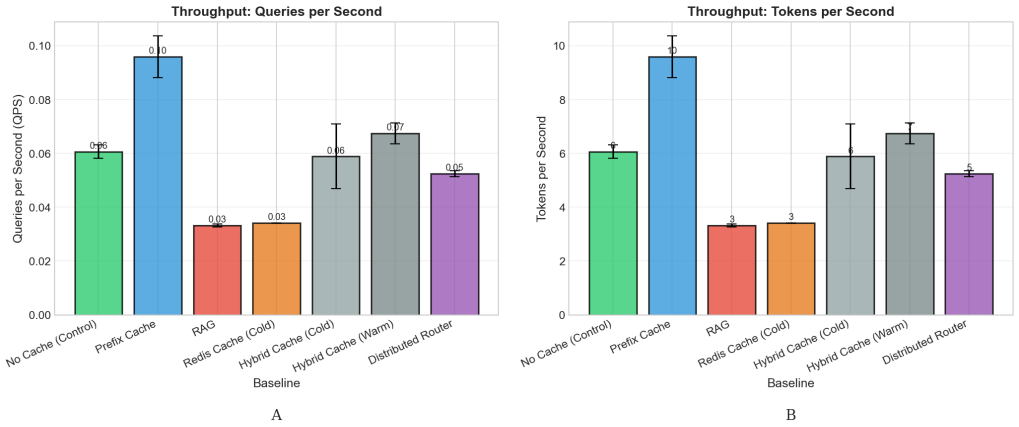


**Figure 3. Throughput comparison: (a) QPS and (b) TPS.**

## 4.2. Latency Decomposition and Variance

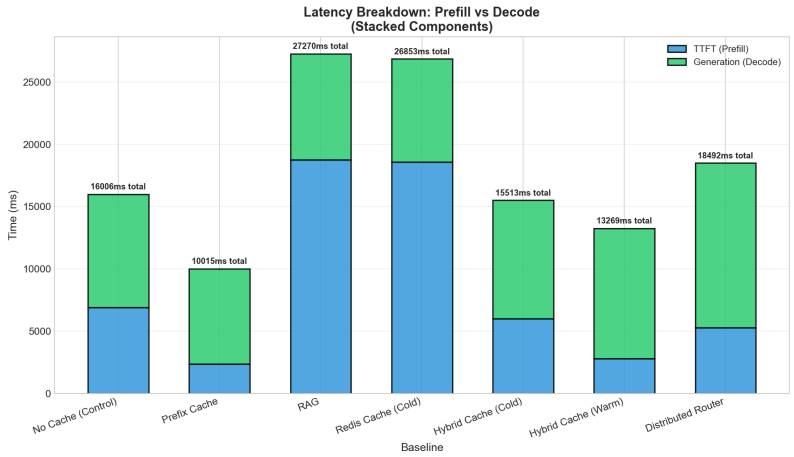


**Figure 4. Latency decomposition into TTFT (prefill) and generation time.**

Figure 4 decomposes total latency into prefill and generation components. The stacked bars confirm that Prefix Cache's advantage is concentrated in the prefill phase, while generation time remains comparable across baselines.

**Table 3. TTFT Distribution: Median vs Tail Latency (ms)**

| Baseline | p50 | p95 | Ratio |
|---|---|---|---|
| No Cache | 6467 | 10,435 | 1.6× |
| Prefix Cache | 2300 | 4716 | 2.1× |
| RAG | 18,871 | 23,895 | 1.3× |
| Redis Cache Cold | 18,629 | 23,974 | 1.3× |
| Hybrid Cache Cold | 5928 | 10,127 | 1.7× |
| Hybrid Cache Warm | 2947 | 4462 | 1.5× |
| Distributed Router Replicated | 2876 | 21,896 | **7.6**× |

Table 3 reveals that Distributed Router has a 7.6× p95/p50 TTFT spread (2876ms median vs 21,896ms tail), caused by cold replicas that have not yet warmed their local

prefix cache. Hybrid Cache Cold shows a similar cold-start effect (±4290 ms TTFT std across trials), while RAG and Redis Cache Cold are consistently slow but stable (1.3× ratio).

### 4.3. Quality Results

Figure 5 shows that gold-context baselines (No Cache, Prefix Cache Warm/Cold, Distributed Router) preserve the strongest faithfulness, while retrieval-backed settings are consistently weaker. This separation is consistent with Chan et al. [2025], who show that retrieved passages can weaken factual grounding when the gold context is already available.

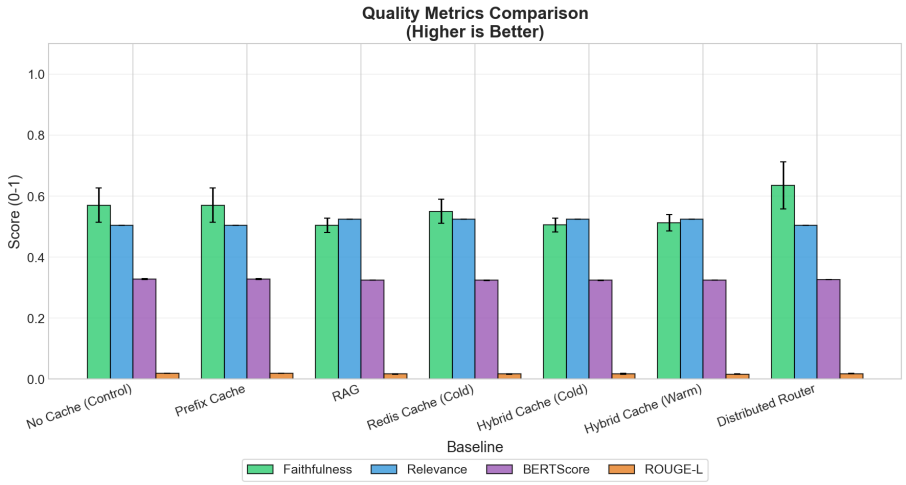


**Figure 5. Quality metrics comparison across baselines.**

Table 4 quantifies these differences. Prefix Cache has the lowest latency (Table 2) but identical faithfulness to No Cache (0.570), confirming that native prefix caching preserves exact quality parity. Retrieval-based baselines are consistently less faithful (0.504-0.550), with marginally higher relevance (0.525 vs 0.505) that does not compensate for the grounding loss. BERTScore is nearly constant across all baselines (0.324–0.328, with standard deviation below 0.001), and relevance shows no variation within each context group, making faithfulness the most informative quality metric in this setting.

**Table 4. Quality Metrics and Latency-Faithfulness (Mean ± Std)**

| Baseline | Latency (ms) | Faithfulness | BERTScore | Relevance |
|---|---|---|---|---|
| Prefix Cache | 10,015 | 0.570 ± 0.056 | 0.328 | 0.505 |
| No Cache | 16,006 | 0.570 ± 0.056 | 0.328 | 0.505 |
| Hybrid Cache Warm | 13,269 | 0.512 ± 0.027 | 0.324 | 0.525 |
| Hybrid Cache Cold | 15,513 | 0.506 ± 0.022 | 0.324 | 0.525 |
| Distributed Router | 18,492 | **0.636 ± 0.078** | 0.327 | 0.505 |
| Redis Cache Cold | 26,853 | 0.550 ± 0.040 | 0.324 | 0.525 |
| RAG | 27,270 | 0.504 ± 0.023 | 0.324 | 0.525 |

The ceiling faithfulness value of 0.570 on the gold passage is lower than typical raw extraction metrics, because it is an expected artifact of how our strict metric interacts

with the dataset and model characteristics. In CAGE, Faithfulness is computed as a strict NLI-based (Natural Language Inference) metric. It splits the generated answer into discrete claims and computes, for each claim, the mean entailment probability with respect to the context using a cross-encoder.

SQuADv2 contains unanswerable questions where the text deliberately lacks the necessary information, and the smaller Qwen3-4B model tends to add conversational filler to its responses rather than answering concisely. Because the metric grades each model-generated phrase for factual accuracy, this additional conversational text lowers the overall score. Then, when the system is forced to rely on a search engine (RAG) rather than a perfect cache, its faithfulness score drops to 0.504. This 11.6% drop demonstrates that the metric effectively captures the model's increased factual errors when it reads retrieved, imperfect passages. Ultimately, moving from a cached context to a retrieved context directly degrades factual accuracy.

### 4.4. Cache Telemetry

Table 5 shows that retrieval hit rate alone does not explain performance. To understand what drives TTFT improvement, this table combines retrieval signals, prompt-cache ratios, and the resulting TTFT reduction relative to No Cache.

**Table 5. Cache and Retrieval Telemetry with TTFT Impact**

| Baseline | Retr. Hit | Prompt Cached | Cache Hit (%) | TTFT Reduction (%) |
|---|---|---|---|---|
| No Cache | — | — | 0 | 0 (baseline) |
| Prefix Cache | — | 68.4% | 94 | +65.7 |
| RAG | 98% | — | 0 | −171.4 |
| Redis Cache Cold | 98% | — | 0 | −168.5 |
| Hybrid Cache Cold | 98% | 75.6% | 92 | +13.3 |
| Hybrid Cache Warm | 98% | 89.2% | 100 | +59.7 |
| Distributed Router Replicated | — | 68.4% | 94 | +23.7 |

The retrieval-backed baselines all achieve 98% retrieval hit rate, yet RAG and Redis Cache Cold remain the slowest settings with the largest TTFT increases. Both RAG and Redis Cache Cold have 0% cache-hit rate (retrieval succeeded, but results were not reused), whereas Hybrid Cache Warm reached 100% (retrieval was both correct and reused). Prefix Cache and Distributed Router Replicated report 68.4% cached prompt ratio, Hybrid Cache Cold 75.6%, and Hybrid Cache Warm 89.2%. Prompt-cache ratios are computed over requests with vLLM prompt-usage telemetry (coverage: 47/50 for Prefix Cache and Distributed Router, 46/50 for Hybrid Cold, 50/50 for Hybrid Warm).

Figure 6 Baselines cluster into two groups: those with high local cache-hit rates (Prefix Cache, Hybrid Warm, Distributed Router, Hybrid Cold) appear in the upper-right quadrant with positive TTFT reductions, while those without local reuse (RAG, Redis Cache Cold) fall to the lower-left with large TTFT penalties. This confirms that local prompt-cache reuse, not retrieval success, is the primary driver of TTFT improvement.

These signals demonstrate why this framework records cache and retrieval metrics alongside serving performance. Without cache hits and prompt-cached columns, a system reporting a 98% retrieval hit rate would appear healthy, but RAG and Redis Cache Cold show that retrieval success does not translate into serving efficiency.

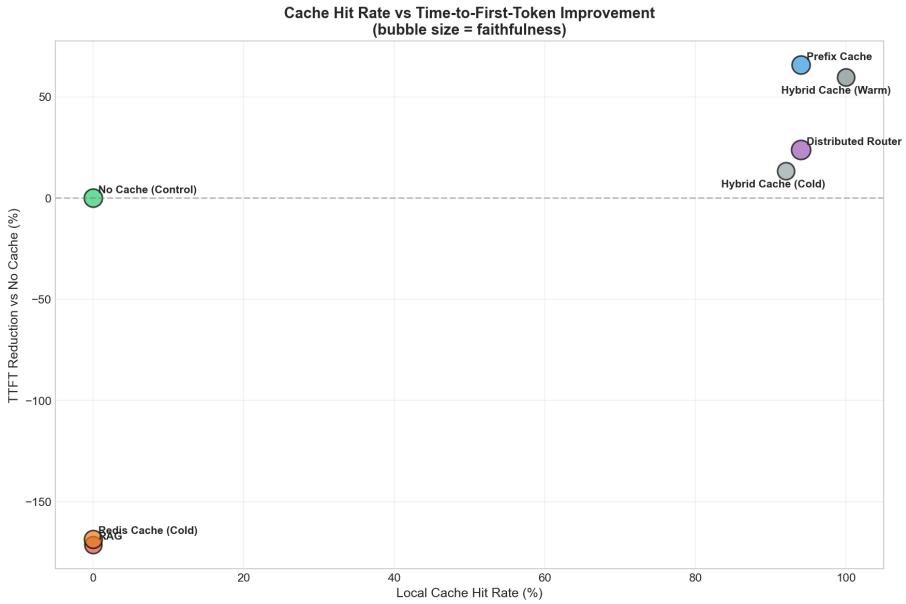


**Figure 6. Local cache-hit rate versus TTFT reduction relative to No Cache.**

## 5. Conclusion

This paper presented CAGE, to our knowledge, one of the first comprehensive frameworks for evaluating Cache-Augmented Generation systems. The objective was to address the lack of a unified methodology for assessing cache-aware LLM systems, especially in settings where latency, correctness, and operational reliability must be considered together. Our initial results were designed as a controlled validation step, comparing baselines under common experimental conditions.

Our results (Tables 2 and 4) show that native prefix caching was the strongest baseline, reducing latency by 37.4% and TTFT by 65.7% while preserving identical faithfulness (0.570). Tail latency analysis (Table 3) further revealed that mean metrics alone are insufficient, as the Distributed Router baseline showed a 7.6× p95/p50 TTFT spread. The Distributed Router baseline improved responsiveness (Table 2), but the benefit is concentrated in the early response stage; because TPOT remained high, end-to-end latency did not become the best result.

Future work will extend this proposal by transitioning the architecture to cloud-provisioned GPUs to evaluate prefix caching under authentic KV-cache memory pressure. We will also focus on distributed high-performance computing (HPC) validation by deploying a multi-node GPU cluster on Google Cloud Platform at a larger scale. Finally, we will evaluate real cross-node KV cache transfer costs, speculative decoding interactions, and native telemetry in the context of vLLM disaggregated prefilling.

## References

Souza, W. J., Marques-Neto, H. T., and Freitas, H. C., Retrieval-Augmented Large Language Models for Computer Architecture Learning and Design Assistance. In *International Journal of Computer Architecture Education (IJCAE)*, vol. 14, no. 1, pages. 12—18, 2025.

Lewis, P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Chan, B. J., Chen, C.-T., Cheng, J.-H., and Huang, H.-H., Don't Do RAG: When Cache-Augmented Generation is All You Need for Knowledge Tasks. In *WWW (Companion Volume) 2025*, pages 893–897, 2025.

Lu, S., Wang, H., Rong, Y., Chen, Z., and Tang, Y., TurboRAG: Accelerating Retrieval-Augmented Generation with Precomputed KV Caches for Chunked Text. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 6588–6601, 2025.

Es, S., James, J., Espinosa Anke, L., and Schockaert, S., RAGAs: Automated Evaluation of Retrieval Augmented Generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158, 2024.

Friel, R. et al. RAGBench: Explainable Benchmark for Retrieval-Augmented Generation Systems. *arXiv preprint arXiv:2407.11005*, 2024.

Saad-Falcon, J., Khattab, O., Potts, C., and Zaharia, M., ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1)*, pages 338–354, 2024.

Rau, D., Déjean, H., Chirkova, N., Formal, T., Wang, S., Clinchant, S., and Nikoulina, V., BERGEN: A Benchmarking Library for Retrieval-Augmented Generation. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 7640–7663, 2024.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I., Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

Li, H. et al. A Survey on Large Language Model Acceleration based on KV Cache Management. *arXiv preprint arXiv:2412.19442*, 2024.

Javidnia, N., Rouhani, B. D., and Koushanfar, F., Key, Value, Compress: A Systematic Exploration of KV Cache Compression Techniques. In *IEEE CICC 2025*, 2025.

Qin, R. et al., Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving. *arXiv:2407.00079*, 2024.

Sahoo, P., Meharia, P., Ghosh, A., Saha, S., Jain, V., and Chadha, A., A Comprehensive Survey of Hallucination in Large Language, Image, Video and Audio Foundation Models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 11709–11724, 2024.

Barnett, S., Kurniawan, S., Thudumu, S., Brannelly, Z., and Abdelrazek, M., Seven Failure Points When Engineering a Retrieval Augmented Generation System. In *CAIN 2024*, pages 194–199, 2024.

Zhong, Y. et al. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024.

Li, Z., Li, C., Zhang, M., Mei, Q., and Bendersky, M., Retrieval Augmented Generation or Long-Context LLMs? A Comprehensive Study and Hybrid Approach. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 881–893, 2024.

Redis. *Redis Official Documentation*, 2025. Available at: `https://redis.io/docs`.