

RAGPULSE: A Real-World RAG Workload Capturing Authentic Temporal Reuse Patterns for Efficient Serving

Zhengchao Wang and Yitao Hu and Zhuxuan Chang and Jianing Ye
Guotao Yang and Tianhao Lai and Jiazheng Yu and Zhixin Zhao
Liang Zheng and Youpeng Deng and Sheng Chen and Keqiu Li
Tianjin University, China

Abstract

Retrieval-augmented generation (RAG) has emerged as a critical paradigm for deploying reliable and knowledge-intensive large language model (LLM) applications. Its unique execution pipeline introduces large overhead in system retrieval, such as massive context loading and document dependencies. To reduce this overhead, existing work on RAG serving systems often relies on generic LLM inference traces or synthetic RAG datasets. However, these approaches fail to capture authentic temporal reuse patterns of RAG workloads, resulting in a disconnect between theoretical optimization and real-world deployment efficiency.

To bridge this gap, this paper introduces RAGPULSE, an open-source RAG workload trace dataset that preserves authentic temporal information missing in existing traces. Collected from a production-grade university Q&A system serving over 40,000 students and faculty members since April 2024, the dataset comprises 134K requests over 389 days. It unveils key RAG dynamics including time-sensitive document reuse, skewed access patterns over long time scales, and bursty temporal locality. By capturing these temporal reuse patterns, RAGPULSE empowers the systems community to develop and validate next-generation optimization techniques, offering practical guidance for efficient RAG serving. The trace is available at <https://github.com/flashserve/RAGPulse>.

1 Introduction

Large language models (LLMs) (Huang et al., 2025; Qin et al., 2025; Xiao et al., 2025) face persistent challenges in production settings, most notably *knowledge cutoff* and *hallucination*. Retrieval-augmented generation (RAG) (Hsia et al., 2025; Lu et al., 2024; Wang et al., 2025a) mitigates these limitations by grounding model generation in external knowledge sources. As illustrated in Figure 1,

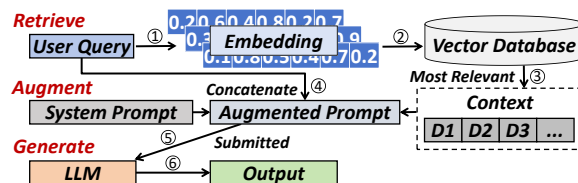


Figure 1: Workflow of RAG.

a RAG request traverses a multi-stage pipeline encompassing retrieval, augmentation, and generation. Today, RAG stands as the dominant paradigm for knowledge-intensive LLM applications.

Although RAG significantly improves answer accuracy, its system-level performance (Jin et al., 2025; Shang et al., 2025; Wang et al., 2025d) remains hindered by the lack of representative traces. Both generic LLM (Wang et al., 2025c; shareAI, 2023; Wu et al., 2025; Yang et al., 2025; Sreeram et al., 2025) and synthetic RAG (Katsis et al., 2025; Qiu et al., 2024; Zhong et al., 2021) datasets miss RAG-specific dynamics. Most synthetic datasets are generated via random independent sampling or fixed-interval arrivals, destroying the bursty, time-varying structure of real user behavior. Thus, three temporal dimensions are missing: bursty arrival patterns, document popularity drift, and multi-turn session continuity. These are central to modern serving optimizations like Position Independent Caching (PIC) (Yao et al., 2025a; Hu et al., 2024), prefix-aware attention (Yi et al., 2026), content-aware scheduling (Qin et al., 2025), and KV cache management (Touzeau et al., 2019; Shah and Siddiqui, 2023). Consequently, systems designed with generic traces show a substantial performance gap in production RAG workloads.

In this paper, we address this gap by presenting RAGPULSE, an open-source RAG workload trace dataset that preserves authentic temporal information missing in existing traces. Through an empirical study, we provide the first systematic characterization of authentic RAG workloads and

demonstrate how their distinctive temporal properties inform specialized serving optimizations. Our contributions are summarized as follows:

- **Workload Characterization (§3).** We release a public RAG trace consisting of 134K real user requests and identify key workload properties, including highly skewed document reuse and bursty temporal locality. These observations are essential for RAG-specific serving system optimization, including content-aware scheduling and KV cache management.
- **System Infrastructure (§4).** We describe the trace collection methodology of TWEN, a large-scale production RAG system that is representative of canonical RAG systems. Relying on RAG techniques, this system has served over 40,000 students over 389 days.
- **Benchmarking Framework (§5).** We build an end-to-end evaluation platform that supports time-calibrated execution. This enables systematic measurement of RAG serving efficiency using industry-standard metrics, including time-to-first-token (TTFT) and time-per-output-token (TPOT).
- **System Bottleneck Validation (§6).** Using RAGPULSE, we reveal RAG-specific system bottlenecks that are obscured by generic LLM traces and synthetic RAG datasets, including content-aware scheduling policy and cache management policy. These findings motivate the need for RAG-aware serving and tiered caching architectures.

To the best of our knowledge, RAGPULSE is the first open-source, production-grade trace designed specifically for RAG serving. By grounding system design and evaluation in real-world RAG workloads, our work enables more specialized, data-driven optimization of LLM serving systems, particularly for long-context inference and memory-efficient serving (Zheng et al., 2026).

2 Preliminary and Motivation

2.1 Why We Need a Real-World RAG Trace with Temporal Information?

Existing generic LLM-Specific traces are fundamentally inadequate for RAG serving systems. This limitation arises from the distinctive workload characteristics of RAG, which diverge substantially from standalone LLM inference and thus require dedicated, RAG-specific traces.

First, RAG introduces an inherent multi-stage

execution pipeline. Each request traverses multiple tightly coupled phases—including retrieval, re-ranking, and generation (Yu et al., 2025; Asai et al., 2024; Lu et al., 2024). End-to-end latency, resource utilization, and cache behavior are therefore governed by cross-stage interactions rather than by generation alone. Generic LLM traces, which primarily capture the prefilling and decoding phase, fail to expose these composite system effects.

Second, RAG performance is highly dependent on both the underlying knowledge base and real-world query patterns (Yue et al., 2025; He et al., 2024; Gutiérrez et al., 2024). Workload dynamics such as query similarity and document cache hit rates vary significantly with user behavior. Moreover, practical deployments often exhibit strong inter-request correlations, where consecutive queries share overlapping contexts or retrieved documents. Accurately capturing these temporal and content-level dependencies is essential for optimizing scheduling, batching, and KV cache reuse policies (Agarwal et al., 2025; Jin et al., 2024).

Third, most existing RAG traces are derived from synthetic or curated chat workloads, which typically contain limited document corpora and lack authentic temporal locality information. Optimizations validated on non-representative traces often fail to generalize to production environments, resulting in performance gaps and unstable system behavior. This disconnect substantially hinders the practical advancement of RAG serving systems.

Beyond characterization, RAG-specific traces are critical for enabling joint optimization of retrieval and generation. Treating these components as independent black boxes limits the design space of RAG serving systems. In contrast, realistic RAG traces make it possible to co-optimize retrieval caching, query-adaptive batching, and scheduling decisions across pipeline stages (Wang et al., 2024).

Taken together, these limitations underscore the need for a real-world, production-grade RAG trace to ground system design and evaluation in realistic workload behaviors.

2.2 What We Can Do with a RAG Trace?

A high-fidelity RAG trace that reflects real deployment workloads enables a broad range of system-level analyses and optimizations that are otherwise inaccessible.

Fine-grained bottleneck analysis. The trace enables precise decomposition of end-to-end RAG latency, quantifying performance contributions and

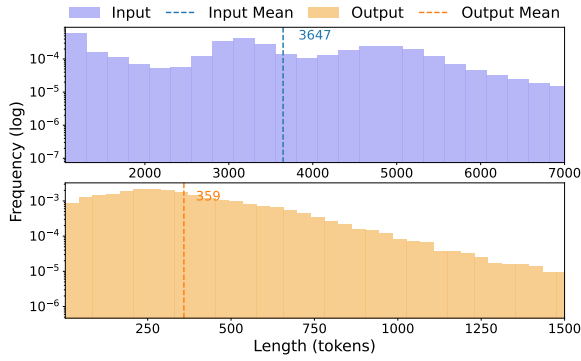


Figure 2: Input and output token length distribution.

bottlenecks of retrieval, re-ranking, and generation under realistic workload conditions.

Informed scheduling and caching design. By capturing inter-request correlations and content reuse patterns, the trace provides a principled foundation for evaluating and designing advanced strategies such as adaptive request batching and KV cache management.

Realistic workload modeling and benchmarking. The trace supports construction of high-fidelity workload models, simulators, and standardized benchmarks, improving the validity and reproducibility of RAG system evaluations.

Understanding emerging RAG application behaviors. It enables the study of how real-world RAG workloads evolve from simple question answering to more complex behaviors, including problem rewriting, multi-turn dialogue, and agent-like task execution.

Collectively, these capabilities position RAG-specific traces as a key enabler for bridging the gap between theoretical RAG system research and real-world deployment.

3 Open-Source RAG Trace: RAGPULSE

To bridge the gap between RAG systems research and real-world deployment, we introduce RAGPULSE, an open-source RAG workload trace collected from a production university-wide policy question-answering system. The service has been continuously operating since April 2024 and serves over 40,000 students and faculty members. We release the core trace of RAGPULSE as a public dataset and commit to its continuous update as the system evolves.

3.1 Data Format and Content

Each record in the RAGPULSE trace captures key system-level runtime information for a RAG re-

1	{
2	"timestamp": "559285",
3	"input_length": 3861,
4	"output_length": 127,
5	"hash_ids": {
6	"sys_prompt": [8325, 8326,
7	11575],
8	"passages_ids": [6123, 7239,
9	6124, 1167, 7250, 5448],
10	"history": [15215],
11	"web_search": [20319, 20320],
12	"user_input": [23648]
13	},
	"session_id": "1758081660427-
	xa8rbsd2uco1"
	}

Table 1: Request component sample.

1	{ "hash_id": 7250, "length": 328 }
2	{ "hash_id": 8325, "length": 512 }
3	{ "hash_id": 20319, "length": 484 }

Table 2: Token length per component.

quest. The dataset we are releasing at this time covers 134K request records over 389 days. As shown in Figure 2, the input and output token lengths of the RAGPULSE workload demonstrate a clear concentration trend. Specifically, input token lengths primarily cluster around 3000 tokens, while output tokens are concentrated around 300 tokens. Furthermore, as detailed in Table 1, each record in the main trace file contains the following key fields.

- **timestamp**: request arrival time in seconds relative to the start of the trace;
- **input_length**: total input token length, aggregating all request components;
- **output_length**: total output token length;
- **hash_ids**: identifiers for all input components, each mapping to a unique document chunk, with token lengths provided as shown in Table 2;
- **session_id**: conversation session identifier.
- **latency_breakdown**: execution timestamps and latency measurements for each processing stage, including knowledge-base retrieval, reranking, and generation, as shown in Table 3;

3.2 Dataset Features and Statistical Analysis

Our analysis of the RAGPULSE trace reveals four characteristics that are particularly informative for RAG serving system design.

Periodic load variation. The trace records precise request arrival times. As shown in Figure 3, which is derived from the six-week trace, request volume exhibits clear diurnal patterns, with higher load

```

1 {
2   "timestamp": "5259",
3   "stages": [
4     {
5       "name": "knowledge_base_retrieval",
6       "start_ms": 1248,
7       "end_ms": 2374,
8       "latency_ms": 1126
9     },
10    {
11      "name": "reranking",
12      "start_ms": 4037,
13      "end_ms": 5185,
14      "latency_ms": 1148
15    },
16    {
17      "name": "generation",
18      "start_ms": 6266,
19      "end_ms": 21062,
20      "latency_ms": 14796
21    }
22  ],
23  "total_latency_ms": 17070
24 }

```

Table 3: Latency breakdown by processing stage.

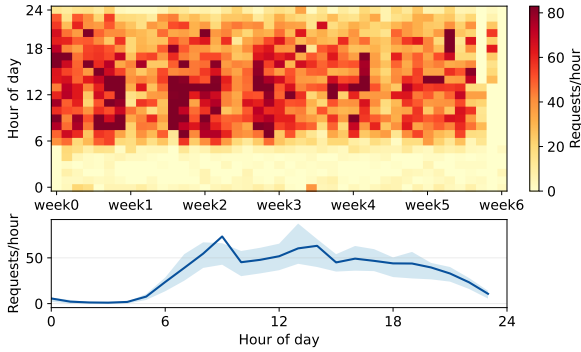


Figure 3: Temporal periodicity in RAGPULSE.

during daytime and significantly reduced activity at night. This behavior closely matches real-world serving workloads (Qin et al., 2025; Wang et al., 2025c) and motivates time-aware scheduling and provisioning strategies.

Dynamic input composition. Figure 4 analyzes the breakdown of input tokens by component. For short requests (below 5,000 tokens), system prompts dominate, contributing over 40% of tokens. As total input length exceeds 3,000 tokens, retrieved passages and dialogue history grow substantially, together accounting for nearly 50% of the input. This observation highlights that context-processing overhead in RAG systems is highly request-dependent rather than uniform.

Skewed document reuse. Analysis of hash_ids reveals a highly skewed access pattern for retrieved documents. As shown in Figure 5(a), a small frac-

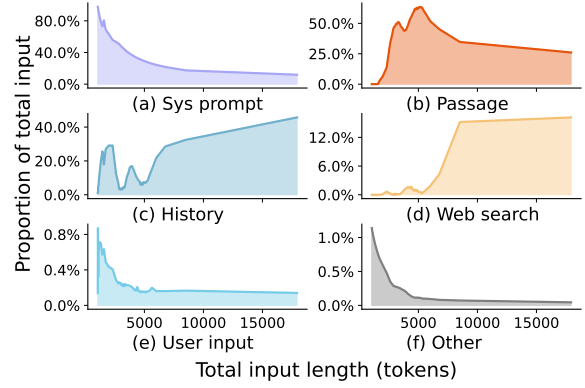


Figure 4: Input token breakdown.

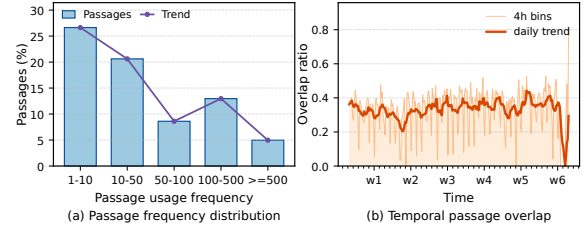


Figure 5: Unique characteristics of RAG traces.

tion of knowledge chunks is repeatedly accessed by many requests, while most chunks are rarely used, forming a pronounced long-tail distribution. This skew strongly motivates retrieval caching and document-aware memory management.

High inter-request document overlap. Temporal analysis further shows substantial overlap in retrieved documents among concurrently arriving requests. As illustrated in Figure 5(b), which is derived from the six-week trace, requests within short time windows frequently reference identical or highly similar knowledge chunks. This property creates significant opportunities for content-aware batching and high-reuse KV cache placement for frequently accessed documents (Jie et al., 2025; Willette et al., 2025; Yao et al., 2025b).

4 Data Source: TWEN System

The RAGPULSE trace dataset is sampled from TWEN, a university-wide Q&A RAG system that has served over 40,000 students and faculties since April 2024. This section details TWEN’s system architecture to provide context for how the trace data was collected.

4.1 System Overview

The core design philosophy of TWEN is the separation of retrieval and generation. To achieve this, TWEN adopts a Python-based microservice

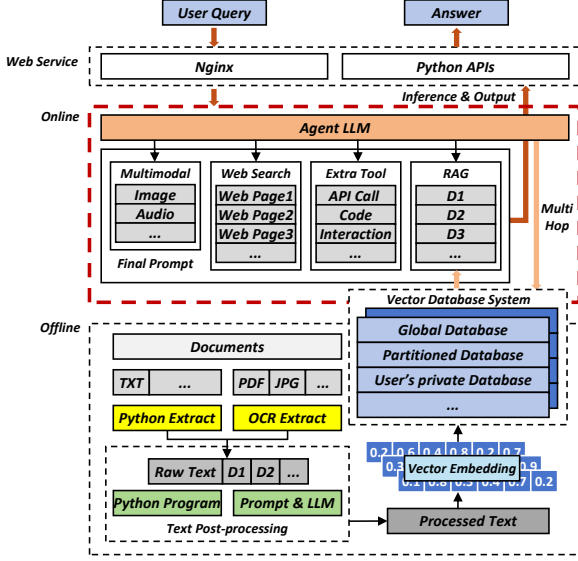


Figure 6: TWEN’s system architecture.

architecture powered by LangChain (Brace Sproul, 2024) and Docker to decouple the CPU-intensive retrieval tasks from the GPU-intensive generation tasks. The architecture is illustrated in Figure 6.

- Generator model: We use Qwen3-235B-A22B model (Bai et al., 2023) for main Q&A and agent interaction functions.
- Embedding model: We use the deployed open-source model infgrad/stella-large-zh-v3-1792d (Infgrad, 2024) for vectorizing all knowledge base documents.
- Orchestration: LangChain (Brace Sproul, 2024) is used to build and orchestrate retrieval flows, prompt templates, and model chains.
- Vector database: Qdrant (Qdr, 2025) is used to store and index knowledge base vectors with a tag-based structure, organized into global, tagged, and user-personal databases.
- Document maintenance: The system uses Python programs and LLM for content extraction, format standardization, and automatic segmentation of files.

4.2 Offline Stage

As shown in Figure 7, the core task of the offline stage is to construct the high-quality vector knowledge base. This stage involves a pipeline that transforms raw various documents into structured embeddings suitable for efficient retrieval.

The knowledge base corpus of university policies and information is acquired via three channels: (1) Manual submission by faculty and department leaders; (2) An upload system for authorized ad-

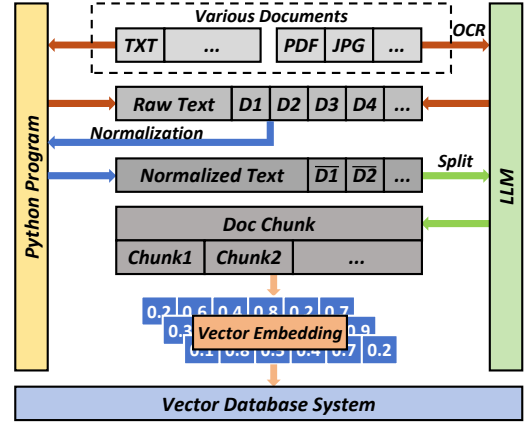


Figure 7: Offline architecture in TWEN.

ministrators; and (3) Web crawlers that periodically pull data from public department websites.

Based on their storage formats, the raw documents are categorized into two types: the first consists of files like TXT, from which content can be easily extracted by Python programs; the second includes files like JPG, where content is not explicitly stored and requires extraction using LLM-based OCR (Lee, 2025). The extracted raw content often has messy formatting. To address this, we develop a specialized agent for text normalization. The normalized text is then intelligently segmented using another dedicated LLM¹ to ensure semantic integrity, resulting in high-quality corpus chunks suitable for building the vector database.

4.3 Online Stage

As shown in Figure 8, the online stage handles user requests and generates the RAGPULSE trace records in the process.

We have developed our agentic RAG system, a comprehensive framework where a central large language model, which we refer to as the Agent LLM, acts as an autonomous controller. When a request arrives, this Agent LLM autonomously plans the optimal processing strategy based on user information and prompts. It can interpret queries from multimodal perspectives (images, audio, text, etc.) and dynamically orchestrates a suite of tools, including web search, RAG, and a series of pre-defined external tools (API calls, programming, etc.) to comprehensively solve problems. Furthermore, it supports proactive self-reflection: the Agent LLM first assesses its own response, and

¹We use a variety of LLM models ranging from 8B to 235B for different sub-tasks such as problem rewriting for cost efficiency.

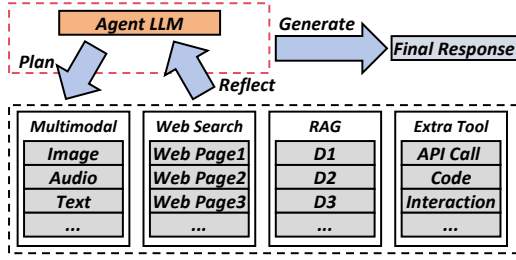


Figure 8: Online architecture in TWEN.

if it deems the current answer insufficient, it will employ various tools in a multi-hop manner until reaching the final response.

The system calls Qwen3-235B model asynchronously in streaming mode using LangChain’s AsyncIteratorCallbackHandler. During this call, the Langfuse (Clemo, 2024) CallbackHandler is activated, capturing the call metadata, including latency, token lengths, user id, session id, and retrieved chunk ids. This captured data is sent to Langfuse database, forming RAGPULSE trace dataset analyzed in this study.

5 Dataset Usage and Applications

To validate the effectiveness of proposed RAG-PULSE dataset for evaluating system performance in real-world environments, we have constructed a dedicated benchmarking platform. This section provides a detailed description of this platform’s specific implementation, the composition and usage methodology of the dataset, and the performance evaluation metrics employed.

5.1 Implementation

We constructed a prototype system for an online RAG service to simulate real-world applications. The system’s core is built upon vLLM (v0.10.1) and LMCACHE (v0.3.1) (Kwon et al., 2023; Cheng et al., 2024; Liu et al., 2024; Yao et al., 2025a), a widely adopted open-source inference framework. To emulate a typical RAG workflow, the system integrates an end-to-end pipeline, encompassing user request reception, text processing, and final result generation. All experiments are conducted on a server equipped with a single NVIDIA A100 GPU with 80GB VRAM. We deploy Qwen3-8B as the generator component of RAG pipeline.

5.2 Scaling RAGPULSE to Any Scale

RAGPULSE’s trace data is characterized by a significant temporal span. This property provides researchers with high flexibility, allowing them to

slice or scale the trace data according to specific experimental goals. For example, a high-traffic period (e.g., one hour) can be isolated to benchmark the system’s peak concurrent processing capabilities. Conversely, the entire long-term trace can be used to assess the long-term efficacy of caching policies, such as the KV cache management. Moreover, an analysis of throughput fluctuations in the trace enables a deeper comprehension of the system’s behavioral patterns under varying loads.

5.3 Performance Metrics

We evaluate the RAG serving system using two standard LLM-serving metrics that capture performance across prefill and decoding phases: (1) Time to First Token (TTFT), measuring end-to-end latency from request arrival to the first generated token and reflecting user-perceived responsiveness; and (2) Time Per Output Token (TPOT), measuring the average latency of subsequent tokens and reflecting generation throughput. This distinction is vital as prefill is compute-bound, whereas decoding performance is primarily limited by memory bandwidth (Agrawal et al., 2024; Qin et al., 2025).

5.4 Demo Use

The RAGPULSE benchmark employs a client-server architecture for workload replay. The server, built on vLLM, handles inference tasks via an OpenAI-compatible API, supporting both streaming and non-streaming modes to capture key metrics such as TTFT and TPOT. An asynchronous, concurrent design ensures stability and high throughput under heavy load.

The client acts as the load generator. Its three-stage workflow includes: (1) preprocessing the trace file; (2) dispatching requests strictly adhering to original timestamps, with an optional time-scaling factor to adjust replay speed; and (3) recording performance metrics for subsequent analysis.

6 System Optimization Validation

While §5.1 demonstrates the basic integration of RAGPULSE in a prototype environment, this section provides a large-scale performance validation on authentic temporal traces to expose system behaviors that general-purpose traces fail to capture.

The experimental framework scales from the initial prototype to a high-performance cluster, utilizing eight NVIDIA A100 GPUs. Unlike the single-instance setup in our application examples, we deploy multiple Qwen3-8B instances to simulate a

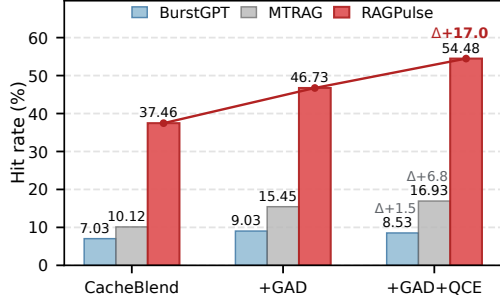


Figure 9: Hit rate across datasets.

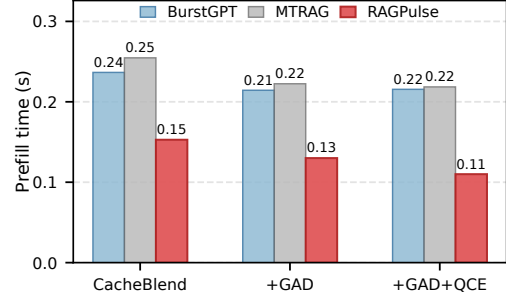


Figure 10: Prefill time across datasets.

high-concurrency production environment. To rigorously stress-test system’s capability of handling long-context requests identified in RAGPULSE, we implement a strict memory partitioning strategy: 30 GB of VRAM per GPU is explicitly reserved for a dedicated KV cache pool. This setup allows us to quantitatively evaluate the storage-compute trade-offs and memory sensitivity under intensive, authentic RAG retrieval scenarios.

We compare RAGPULSE against two representative baselines. The first is BurstGPT (Wang et al., 2025c), a generic LLM inference trace. The second is MTRAG (Katsis et al., 2025), a synthetic RAG dataset. To ensure fair comparison we select approximately 6,000 requests with around 5K input tokens from all three datasets. This length is chosen because it represents a common operating region where RAG and non RAG workloads overlap, enabling clean separation of system performance from input length effects. All experiments are conducted on the open source LMCACHE RAG serving framework (Yao et al., 2025a; Cheng et al., 2024).

6.1 Anatomy of Dataset Characteristics

We first analyze the intrinsic properties of the three datasets. As shown in Figure 9 and Figure 10, the prefill time and cache hit rate exhibit stark differences across datasets. Under the CacheBlend policy alone, RAGPULSE achieves a hit rate of 37.46%, which is substantially higher than BurstGPT’s 7.03% and MTRAG’s 10.12%. This discrepancy stems from fundamental differences in dataset characteristics. BurstGPT is a generic LLM trace where requests lack structured prompt components, limiting prefix based caching due to its strict exact matching requirement. MTRAG is a synthetic RAG dataset where request arrival patterns are artificially generated, resulting in low document overlap (10.15%) across requests and limited reuse

opportunities despite being RAG oriented. In contrast, RAGPULSE captures temporal reuse patterns from user behavior. Its traces preserve fine grained request arrival times and inter request document affinity, enabling document level KV reuse that neither BurstGPT nor MTRAG can provide.

6.2 Scheduling in PIC Systems

Recent RAG serving systems such as CacheBlend, TurboRAG and EPIC (Lu et al., 2024; Hu et al., 2024) have demonstrated that KV cache can be reused at the document level independent of its absolute position in the input sequence, a technique widely adopted in production RAG deployments. Building on this foundation, we evaluate how scheduling strategies perform in this Position Independent Caching (PIC) enabled serving systems using the three datasets. And we consider two scheduling strategies. The first is hit rate aware allocation which prioritizes caching documents with the highest observed access frequency. The second is Global Aware Dispatch (GAD) which considers both cache hit rate and request level document overlap between dispatching requests. As illustrated in Figure 9 and Figure 10, applying GAD on top of CacheBlend yields a dramatic improvement only for RAGPULSE. Specifically, RAGPULSE’s hit rate increases from 37.46% to 46.73%, while its prefill time decreases by 13.33%. For BurstGPT, the hit rate improves marginally from 7.03% to 9.03%, with prefill time decreasing from 0.24 to 0.21. For MTRAG, the hit rate increases from 10.12% to 15.45%, with prefill time decreasing from 0.25 to 0.22. The limited improvement on BurstGPT and MTRAG reflects their inherent lack of authentic document overlap. BurstGPT has no document retrieval mechanism, while MTRAG’s synthetic nature produces documents that are independent and lack temporal correlations. Only RAGPULSE captures real time sequence based doc-

ument overlap across requests, enabling GAD to effectively anticipate and dispatch requests that share document contexts. These results demonstrate that authentic temporal workload is essential for evaluating and optimizing PIC scheduling strategies.

6.3 Cache Policy Sensitivity Analysis

In RAG serving, requests are queued before processing. Can cache eviction policies leverage the queue’s pending requests to improve performance? A natural hypothesis is that a cache eviction policy considering pending requests’ document needs could outperform temporal policies. To test this, we evaluate Queue Aware Cache Eviction (QCE), which uses queue order to prioritize KV pairs likely reused by upcoming requests.

When combining GAD with QCE, as shown in Figure 9 and Figure 10, RAGPULSE continues to show significant gains while the other datasets plateau. RAGPULSE’s hit rate further increases from 46.73% to 54.48%, an additional gain of 7.75 percentage points and a 17.02 percentage point absolute improvement from the CacheBlend baseline of 37.46%. Its prefill time drops from 0.13 to 0.11, a reduction of 15.38%. For BurstGPT and MTRAG, the hit rate changes from 9.03% to 8.53% and from 15.45% to 16.93%, respectively, while prefill time remains nearly unchanged.

The striking difference stems from RAG’s request queue dynamics combined with authentic temporal locality. Unlike the native LRU of CacheBlend, QCE leverages queue level demand to retain document KV pairs that are contextually relevant to requests waiting in the queue. In production RAG deployments, requests arriving within short time windows often share overlapping document needs, creating queue level reuse hotspots that persist within specific temporal windows. RAGPULSE captures this phenomenon through its authentic temporal sequence data, where requests with similar document needs naturally cluster in time. BurstGPT lacks any document overlap, while MTRAG’s synthetic documents are independent and lack temporal clustering, rendering QCE ineffective. The meaningful improvement only on RAGPULSE confirms that authentic temporal queue dynamics is crucial for RAG cache policy design.

6.4 Serving Configuration Sensitivity Analysis

To assess whether the benefits of RAGPULSE generalize beyond a single serving configuration, we

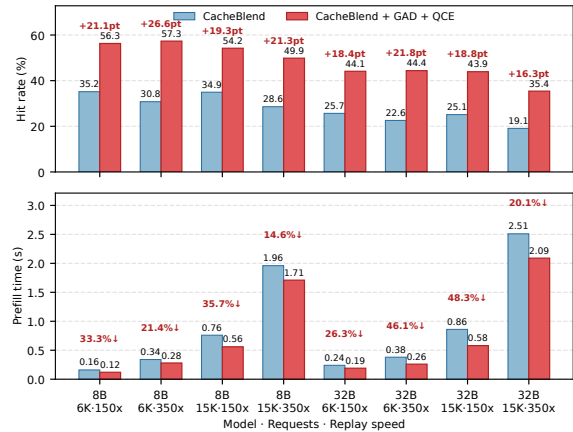


Figure 11: Scalability sensitivity analysis.

vary model size, workload scale, and replay speed. Specifically, we evaluate Qwen3-8B and Qwen3-32B under 6K and 15K request workloads, replayed at 150× and 350× speed. Figure 11 compares CacheBlend against CacheBlend enhanced with GAD and QCE in terms of cache hit rate and Prefill time.

Across all eight configurations, GAD and QCE consistently improve cache efficiency and latency. The cache hit rate increases by 16.32 to 26.55 percentage points, while Prefill time is reduced by 14.60% to 48.27%. These gains persist for both model sizes and workload scales, indicating that RAGPULSE’s authentic temporal order enables GAD and QCE to exploit pending requests and short-window document reuse robustly. Higher replay speeds increase queueing pressure, which can reduce the relative Prefill time improvement in some settings. However, they also expose more pending requests and thus preserve substantial cache-hit-rate gains. Although the 32B model incurs greater KV-cache pressure and higher absolute latency, the improvement remains consistent, demonstrating that the effectiveness of queue-aware cache management is not tied to a specific serving configuration.

6.5 Discussion and Broader Implications

The four sets of experiments consistently show that BurstGPT and MTRAG fail to expose the real optimization opportunities and challenges present in production RAG systems. Generic LLM traces lack document level reuse and temporal coupling. Synthetic RAG datasets while RAG aware are artifactually constrained by the absence of authentic temporal arrival patterns and semantically independent document distributions. Both lead researchers

to overestimate or underestimate the effectiveness of various system optimizations.

We view this work as a starting point rather than a final answer. RAGPULSE is the first to capture authentic temporal reuse patterns from a production deployment spanning hundreds of days and tens of thousands of real users. We hope the dataset serves as a foundation for the community to develop and rigorously evaluate next generation RAG system optimizations including but not limited to temporal aware scheduling and semantic aware caching. By grounding system design in realistic workload characteristics RAGPULSE enables a shift from heuristic driven optimization to data driven system engineering for efficient RAG serving.

7 Related Work

General LLM Serving Traces. General traces (Wang et al., 2025c; shareAI, 2023; Wu et al., 2025) characterize request arrival and traffic bursts to optimize dispatching and resource scaling. However, these datasets focus on dialogue-based workloads and do not capture the document reuse or the data dependencies inherent in RAG serving. Consequently, these coarse-grained profiles lack the structured metadata required to model multi-component RAG requests.

RAG Datasets and Accuracy Benchmarks. Existing RAG benchmarks (Hsia et al., 2025; Yang et al., 2025; Joren et al., 2025) focus on retrieval precision and generation correctness rather than system metrics like KV cache residency. Their static nature ignores dynamic request arrival patterns, limiting their ability to guide infrastructure level performance tuning.

Optimizations for RAG Serving Systems. Recent RAG optimizations focus on KV cache management (Wang et al., 2025a; Lu et al., 2024; Jin et al., 2024), speculative execution (Wang et al., 2025d), and retrieval aware scheduling (Shang et al., 2025; Wang et al., 2025b). Despite these advancements, evaluations rely on synthetic workloads that overlook real world complexities. RAGPULSE fills this gap by providing a production scale trace to validate such systems under authentic, knowledge intensive dynamics.

8 Conclusion

RAG has emerged as the standard paradigm for building trustworthy, high-performance LLM applications. However, its unique retrieval architecture

introduces unprecedented challenges for system optimization. Existing generic LLM traces and synthetic RAG datasets fail to capture RAG specific workload dynamics, creating a significant gap between academic research and deployment.

To bridge this gap, we introduce RAGPULSE, an open-source RAG workload trace dataset that preserves authentic temporal information missing in existing traces. Through in depth analysis, we reveal critical characteristics of production RAG workloads including time sensitive document reuse and bursty temporal locality. These findings provide empirical grounding for next generation RAG optimizations such as temporal aware scheduling and semantic aware caching, establishing a standard for efficient RAG serving.

Limitations

Although RAGPULSE captures authentic temporal reuse patterns, several aspects remain unexplored. First, while the trace includes multi turn conversations, it does not yet capture more complex agentic workflows such as tool use or iterative retrieval generation cycles across multiple reasoning steps. Second, the trace includes only text based documents, and multimodal RAG scenarios involving images or audio are not covered. Finally, to protect user privacy, we release RAGPULSE as an encrypted, privacy-preserving trace. Although this does not affect RAG system optimization, it removes raw semantic information. We will explore incorporating richer semantic metadata through privacy-preserving techniques (Feyisetan and Kasisviswanathan, 2021; Meehan et al., 2022) and consider releasing original content after obtaining user consent in future versions of RAGPULSE.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 62572341). The authors would also like to thank the anonymous reviewers for their valuable comments and constructive suggestions, which helped improve this work.

References

- 2025. Qdrant. <https://qdrant.org.cn/>.
- Shubham Agarwal, Sai Narayan Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak

- Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Kumar Saini. 2025. [Cache-craft: Managing chunk-caches for efficient retrieval-augmented generation](#). *Proceedings of the ACM on Management of Data*, 3:1 – 28.
- Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. [Taming throughput-latency tradeoff in LLM inference with sarathi-serve](#). In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, pages 117–134. USENIX Association.
- Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. [Self-rag: Learning to retrieve, generate, and critique through self-reflection](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.
- Christian Bromann Brace Sproul. 2024. Langchain. <https://python.langchain.com/docs/introduction/>.
- Yihua Cheng, Kuntai Du, Jiayi Yao, and Junchen Jiang. 2024. Do large language models need a content delivery network? *arXiv preprint arXiv:2409.13761*.
- Lotte Verheyden Clemo. 2024. Langchain. <https://github.com/langfuse/langfuse/>.
- Oluwaseyi Feyisetan and Shiva Kasiviswanathan. 2021. [Private release of text embedding vectors](#). In *Proceedings of the First Workshop on Trustworthy Natural Language Processing*, pages 15–27, Online. Association for Computational Linguistics.
- Bernal Jimenez Gutiérrez, Yiheng Shu, Yijia Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. In *NeurIPS*.
- Xianglin He, Yixuan Tian, Yizhou Sun, Nitesh V Chawla, Thomas Laurent, Yann LeCun, Xavier Breson, and Bryan Hooi. 2024. G-rag: Graph retrieval-augmented generation for textual graph understanding and question answering. In *NeurIPS*.
- Jennifer Hsia, Afreen Shaikh, Zora Zhiruo Wang, and Graham Neubig. 2025. [RAGGED: Towards informed design of scalable and stable RAG systems](#). In *Forty-second International Conference on Machine Learning*.
- Junhao Hu, Wenrui Huang, Weidong Wang, Haoying Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. 2024. [Epic: Efficient position-independent caching for serving large language models](#). In *International Conference on Machine Learning*.
- Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinhao Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and XIAOJUAN QI. 2025. [Slim-LLM: Saliency-driven mixed-precision quantization for large language models](#). In *Forty-second International Conference on Machine Learning*.
- Infgrad. 2024. stella-base-zh-v3-1792d. <https://huggingface.co/infgrad/stella-base-zh-v3-1792d/>.
- Shibo Jie, Yehui Tang, Kai Han, Zhi-Hong Deng, and Jing Han. 2025. [Specache: Speculative key-value caching for efficient generation of LLMs](#). In *Forty-second International Conference on Machine Learning*.
- Bowen Jin, Jinsung Yoon, Jiawei Han, and Sercan O Arik. 2025. [Long-context LLMs meet RAG: Overcoming challenges for long inputs in RAG](#). In *The Thirteenth International Conference on Learning Representations*.
- Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. 2024. Raggache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457*.
- Hailey Joren, Jianyi Zhang, Chun-Sung Ferng, Da-Cheng Juan, Ankur Taly, and Cyrus Rashtchian. 2025. [Sufficient context: A new lens on retrieval augmented generation systems](#). In *The Thirteenth International Conference on Learning Representations*.
- Yannis Katsis, Sara Rosenthal, Kshitij Fadnis, Chulaka Gunasekara, Young-Suk Lee, Lucian Popa, Vraj Shah, Huaiyu Zhu, Danish Contractor, and Marina Danilevsky. 2025. Mtrag: A multi-turn conversational benchmark for evaluating retrieval-augmented generation systems. *arXiv preprint arXiv:2501.03468*.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- Matthias A. Lee. 2025. pytesseract. <https://github.com/madmaze/pytesseract/>.
- Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2024. [Cachegen: KV cache compression and streaming for fast large language model serving](#). In *Proceedings of the ACM SIGCOMM 2024 Conference, ACM SIGCOMM 2024, Sydney, NSW, Australia, August 4-8, 2024*, pages 38–56. ACM.

- Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. 2024. Turborag: Accelerating retrieval-augmented generation with precomputed kv caches for chunked text. *arXiv preprint arXiv:2410.07590*.
- Casey Meehan, Khalil Mrini, and Kamalika Chaudhuri. 2022. [Sentence-level privacy for document embeddings](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3367–3380, Dublin, Ireland. Association for Computational Linguistics.
- Ruoyu Qin, Zheming Li, Weiran He, Jiale Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2025. [Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot](#). In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 155–170, Santa Clara, CA. USENIX Association.
- Zexuan Qiu, Jingjing Li, Shijue Huang, Wanjuan Zhong, and Irwin King. 2024. [Clungeval: A chinese benchmark for evaluating long-context large language models](#). *ArXiv*, abs/2403.03514.
- Jainam Shah and Asim Ahmed Siddiqui. 2023. [An improved cache eviction strategy: Combining least recently used and least frequently used policies](#). In *2023 6th International Conference on Advances in Science and Technology (ICAST)*, pages 1–6.
- Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. 2025. [Agentsquare: Automatic LLM agent search in modular design space](#). In *The Thirteenth International Conference on Learning Representations*.
- shareAI. 2023. [Sharegpt-chinese-english-90k bilingual human-machine qa dataset](https://huggingface.co/datasets/shareAI/ShareGPT-Chinese-English-90k). <https://huggingface.co/datasets/shareAI/ShareGPT-Chinese-English-90k>.
- Shiva Sreeram, Alaa Maalouf, Pratyusha Sharma, and Daniela Rus. 2025. Efficient llm adaptation using a single gradient step on 100 samples. In *NeurIPS*.
- Valentin Touzeau, Claire Maïza, David Monniaux, and Jan Reineke. 2019. [Fast and exact analysis for lru caches](#). *Proc. ACM Program. Lang.*, 3(POPL).
- Wenbin Wang, Yongcheng Jing, Liang Ding, Yingjie Wang, Li Shen, Yong Luo, Bo Du, and Dacheng Tao. 2025a. [Retrieval-augmented perception: High-resolution image perception meets visual RAG](#). In *Forty-second International Conference on Machine Learning*.
- Xingliang Wang, Zemin Liu, Junxiao Han, and Shuiguang Deng. 2025b. [Rag4gfm: Bridging knowledge gaps in graph foundation models through graph retrieval augmented generation](#). In *NeurIPS*.
- Yuxin Wang, Yuhao Chen, Zeyu Li, Xueze Kang, Yuchuan Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. 2025c. [BurstGPT: A real-world workload dataset to optimize llm serving systems](#). In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25)*, Toronto, ON, Canada. ACM.
- Zheng Wang, Shu Teo, Jieer Ouyang, Yongjun Xu, and Wei Shi. 2024. [M-RAG: Reinforcing large language model performance through retrieval-augmented generation with multiple partitions](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1966–1978, Bangkok, Thailand. Association for Computational Linguistics.
- Zilong Wang, Zifeng Wang, Long Le, Steven Zheng, Swaroop Mishra, Vincent Perot, Yuwei Zhang, Anush Mattapalli, Ankur Taly, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2025d. [Speculative RAG: Enhancing retrieval augmented generation through drafting](#). In *The Thirteenth International Conference on Learning Representations*.
- Jeffrey Willette, Heejun Lee, Youngwan Lee, Myeongjae Jeon, and Sung Ju Hwang. 2025. [Training free exponential context extension via cascading KV cache](#). In *The Thirteenth International Conference on Learning Representations*.
- Zhengxuan Wu, Aryaman Arora, Atticus Geiger, Zheng Wang, Jing Huang, Dan Jurafsky, Christopher D Manning, and Christopher Potts. 2025. [Axbench: Steering LLMs? even simple baselines outperform sparse autoencoders](#). In *Forty-second International Conference on Machine Learning*.
- Yicheng Xiao, Lin Song, Rui Yang, Cheng Cheng, Yixiao Ge, Xiu Li, and Ying Shan. 2025. [LoRA-gen: Specializing large language model via online loRA generation](#). In *Forty-second International Conference on Machine Learning*.
- Qi Yang, Chenghao Zhang, Lubin Fan, Kun Ding, Jieping Ye, and Shiming Xiang. 2025. [Re-ranking reasoning context with tree search makes large vision-language models stronger](#). In *Forty-second International Conference on Machine Learning*.
- Jiayi Yao, Hanchen Li, Yuhao Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2025a. [Cacheblend: Fast large language model serving for RAG with cached knowledge fusion](#). In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pages 94–109. ACM.
- Jinwei Yao, Kaiqi Chen, Kexun Zhang, Jiaxuan You, Binhang Yuan, Zeke Wang, and Tao Lin. 2025b. [DeFT: Decoding with flash tree-attention for efficient tree-structured LLM inference](#). In *The Thirteenth International Conference on Learning Representations*.
- Jinjun Yi, Zhixin Zhao, Yitao Hu, Ke Yan, Weiwei Sun, Hao Wang, Laiping Zhao, Yuhao Zhang, Wenxin Li,

- and Keqiu Li. 2026. [Pat: Accelerating llm decoding via prefix-aware attention with resource efficient multi-tile kernel](#). In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS '26*, page 1396–1412, New York, NY, USA. Association for Computing Machinery.
- Yue Yu, Wei Ping, Zihan Liu, Boxin Wang, Jiaxuan You, Chao Zhang, Mohammad Shoeybi, and Bryan Catanzaro. 2025. Rankrag: unifying context ranking with retrieval-augmented generation in llms. In *Proceedings of the 38th International Conference on Neural Information Processing Systems, NIPS '24*, Red Hook, NY, USA. Curran Associates Inc.
- Zhenrui Yue, Honglei Zhuang, Aijun Bai, Kai Hui, Rolf Jagerman, Hansi Zeng, Zhen Qin, Dong Wang, Xuanhui Wang, and Michael Bendersky. 2025. [Inference scaling for long-context retrieval augmented generation](#). In *The Thirteenth International Conference on Learning Representations*.
- Liang Zheng, Bowen Shi, Yitao Hu, Jiawei Zhang, Ruofan Li, Sheng Chen, Wenxin Li, and Keqiu Li. 2026. [Mosaic: Unlocking long-context inference for diffusion llms via global memory planning and dynamic peak taming](#). *ArXiv*, abs/2601.06562.
- Ming Zhong, Da Yin, Tao Yu, Ahmad Zairi Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir R. Radev. 2021. [Qmsum: A new benchmark for query-based multi-domain meeting summarization](#). In *North American Chapter of the Association for Computational Linguistics*.