RAIS RESEARCH ASSOCIATION for INTERDISCIPLINARY STUDIES
March 12-13, 2026
DOI:10.5281/zenodo.19535775

AI use of Context Augmented Generation (CAG) and Beyond

Atif Farid Mohammad

Capitol Technology University, Laurel, MD, USA
afmohammad@captechu.edu

Abstract: Large language models have significantly improved their ability to handle longer blocks of text, leading to a new method called Context Augmented Generation (CAG), which differs from Retrieval Augmented Generation (RAG) by loading all relevant information into the model's memory beforehand and storing it in a key-value (KV) format before any questions are asked. This research examines how CAG works with a proposed algorithm, including its structure, the role of key-value caching, and its performance compared with Retrieval Augmented Generation in answering general knowledge questions. It also discusses the advantages and disadvantages of each method, potential hybrid approaches, and areas for future research. The findings suggest that CAG can reduce response times, simplify system design, and deliver accurate results for manageable knowledge bases, though it remains constrained by the limits of how much text it can process and the challenges of handling very long contexts.

Keywords: CAG, RAG, Key-Value Cache, Long-Context LLMs, Knowledge-Intensive NLP

1. Introduction

The ability of language models to reason over external knowledge has been one of the most consequential research directions of the past five years. Retrieval Augmented Generation (RAG) (Lewis et al., 2020) established a widely adopted blueprint: at query time, a retrieval engine selects relevant documents from a large corpus, injects them into the prompt, and feeds the augmented context to a generative model. The shared approach separates the model's built-in knowledge from external domain information, allowing smaller, fixed models to work with constantly updated data. However, RAG faces some challenges: each question requires a retrieval process that slows response times, depends on additional infrastructure for embeddings, and risks errors in document selection, which can lead to inaccurate or incomplete answers. As knowledge bases grow, the probability of retrieving marginally relevant or noisy passages increases, imposing an accuracy ceiling that more sophisticated re-ranking and fusion strategies only partially address (Gao et al., 2023).

A confluence of recent advances creates an alternative path. Modern LLMs (Bhatt & Mohammad, 2025; Bhowmick et al., 2026), such as Llama 3.1 (Grattafiori et al., 2024), now support context windows of 128 thousand tokens or more. Equally important, the key-value (KV) caching mechanism inherent to transformer inference (Pope et al., 2023) makes it practical to precompute and persist the model's internal representation of a document set, paying the encoding cost only once and reusing it across many queries. These two advances, long context and persistent KV caches, jointly enable Context Augmented Generation (CAG) (Chan et al., 2025), a retrieval-free paradigm that preloads the entire knowledge base before query time. This paper provides a comprehensive in-depth novel proposition of CAG.

2. Background and Related Work

RAG was introduced by Lewis et al. (2020) as a general framework for knowledge-intensive NLP tasks. A dense retrieval model encodes queries and documents into a shared embedding space; at inference the top-k most similar passages are retrieved and concatenated with the query before generation. Subsequent work has expanded RAG in several directions: sparse BM25 retrieval, late-interaction models (ColBERT), iterative and self-reflective retrieval, graph-based indexing, and adaptive chunking strategies (Gao et al., 2023; Bhatt & Mohammad, 2025). Despite this progress, every RAG variant inherits a fundamental dependency on the retrieval pipeline, a

dependency that introduces both latency and failure modes absent from purely parametric generation.

The context window of transformer-based LLMs (Bhowmick et al., 2026) has expanded by three orders of magnitude since the original GPT-2 release. Benchmarks such as RULER (Hsieh et al., 2024) have systematically probed the effective context length of modern models, revealing that the nominal training context length often overstates practical reasoning capacity models exhibit a "lost in the middle" phenomenon where information positioned far from the beginning or end of a long context is under-attended. Nonetheless, for structured knowledge sources (e.g., FAQs, product documentation, and regulatory texts), the relevant content can typically be compressed to fit within 32K to 128K tokens, making long-context models viable for full-context injection. Transformer inference decomposes into a prefill phase (processing all input tokens in parallel) and an autoregressive decode phase (generating one token per step). During prefill, the attention computation for each layer produces key and value tensors that are stored in a KV cache and reused throughout decoding (Pope et al., 2023). CAG exploits this by precomputing the KV cache for the entire knowledge base offline, then loading it at inference time before appending the user query. This eliminates the prefill overhead for the knowledge-base tokens on every subsequent request, dramatically reducing time-to-first token (TTFT).

3. Context Augmented Generation: Architecture and Pipeline

3.1 Formal Definition

Let M denote a long-context LLM with maximum context length C tokens, and let $D = \{d_1, d_2, \ldots, d_n\}$ be a curated knowledge base satisfying $\sum|d_i| \leq C$. CAG operates in two decoupled phases.

- Phase 1: D is formatted into a prompt template P(D) and passed through M's prefill stack. The resulting KV tensors KV(D) = KV-Encode(P(D)) are serialized to disk. This phase executes once and is amortized across all subsequent queries.
- Phase 2: Given a user query q, the system deserializes KV(D) into GPU memory, appends a tokenized query segment [q] to the already-encoded prefix, and runs only the decode phase of M. No external retrieval step is performed. The generative output depends on the full knowledge base through the preloaded cache. A practical complication arises from multi-turn conversations: the KV cache grows with each dialogue turn. CAG addresses this through a reset mechanism that truncates the cache to the checkpoint saved after Phase 1, before appending the next user query (Chan et al., 2025).

This preserves the full knowledge context across turns while bounding cache memory to $O(C \cdot L \cdot d)$ where L is the number of transformer layers and d is the head dimension. An important refinement of baseline CAG is contextual chunk enrichment, in which each document segment is paired with surrounding context (section headings, document metadata, preceding sentences) before preloading. This enrichment ensures that attention over a given chunk encodes its semantic provenance rather than treating it as an isolated fragment. Enriched CAG consistently yields higher BERTScore on multi-hop question-answering datasets relative to non-enriched variants, with the gain growing as chunk granularity decreases (Rajpurkar et al., 2016).

4. Algorithm: Context-Aware Generation (CAG)

Inputs

- **M**: Long-context LLM with maximum context length **C** tokens.
- **$D = \{d_1, d_2, \ldots, d_n\}$**: Curated knowledge base, where $\sum|d_i| \leq C$.
- **q**: User query.
- **L**: Number of transformer layers in M.
- **d**: Head dimension of M.

Phase 1: Knowledge Preloading (Offline)

1. **Format Knowledge Base:**
   - Construct a prompt template P(D) by concatenating all documents in D:

$$P(D) = \text{concat}(d_1, d_2, \ldots, d_n)$$

   - Optionally, apply contextual chunk enrichment to each d_i:
   - Augment d_i with surrounding context (e.g., section headings, metadata, preceding sentences).
   - Let d_i' denote the enriched chunk.
2. **Encode Knowledge Base:**
   - Pass P(D) through M's prefill stack to compute key-value (KV) tensors:

$$KV(D) = \text{KV-Encode}(P(D))$$

   - Serialize KV(D) to disk.

Phase 2: Retrieval-Free Inference (Online)

1. **Initialize Cache:**
   - Deserialize KV(D) into GPU memory.
2. **Process Query:**
   - Tokenize the user query q to obtain [q].
   - Append [q] to the preloaded prefix KV(D).
3. **Run Decode Phase:**
   - Execute the decode phase of M to generate the response r:

$$r = M(KV(D) \circ [q])$$

   - Return r to the user.

Multi-Turn Conversation Reset Mechanism

1. **After Each Turn:**
   - Truncate the KV cache to the checkpoint saved after Phase 1 (i.e., KV(D)).
   - Append the next user query q' to the reset cache.
2. **Cache Memory Bound:**
   - The KV cache memory is bounded by $O(C \cdot L \cdot d)$.

Contextual Chunk Enrichment Refinement

1. **Enrichment Process:**
   - For each document segment d_i, extract surrounding context (e.g., section headings, metadata, preceding sentences).
   - Concatenate d_i with its context to form d_i'.
2. **Impact on Performance:**
   - Enriched CAG improves BERTScore on multi-hop QA datasets, especially for fine-grained chunks.

Output

- **r**: Generated response to the user query q.

Key Properties

- **Efficiency:** Phase 1 is executed once and amortized across all queries. Phase 2 avoids external retrieval, reducing latency.
- **Scalability:** Cache memory is bounded by $O(C \cdot L \cdot d)$, enabling multi-turn conversations.
- **Performance:** Contextual chunk enrichment improves semantic coherence and QA performance.

Pseudocode

```
def CAG(M, D, q)-:
    # Phase 1-: Knowledge Preloading
    P_D = concatenate(D)
    KV_D = M.prefill(P_D)
    save_to_disk(KV_D)
```

```
        # Phase 2-: Retrieval-Free Inference
        KV_D = load_from_disk(KV_D)
        q_tokenized = tokenize(q)
        r = M.decode(KV_D, q_tokenized)
        return r

    def reset_cache(M, KV_D)-:
        M.cache = KV_D  # Truncate to preloaded checkpoint
```

Advantages:

- Eliminates retrieval latency by preloading knowledge.
- Bounded memory usage for multi-turn conversations.
- Contextual enrichment improves response quality.

Limitations:

- Requires $\sum|d_i| \leq C$.
- Cache reset may lose short-term conversational context.

5. Empirical Comparison with RAG

5.1 Experimental Setup

Chan et al. (2025) provide a controlled comparison of CAG against sparse-retrieval RAG (BM25) and dense-retrieval RAG (DPR) on two standard QA benchmarks: SQuAD (Rajpurkar et al., 2016) and HotPotQA. The backbone model is Llama 3.1 8B Instruct with a 128K context window. Accuracy is measured via BERTScore relative to ground-truth answers; latency is wall-clock time from query receipt to first generated token.

5.2 Accuracy Results

CAG matches or exceeds dense RAG across all tested knowledge-base sizes when the corpus fits comfortably within the context window. On HotPotQA, a multi-hop task that requires synthesizing evidence from multiple passages, CAG outperforms both sparse and dense RAG by statistically significant margins. The result is consistent with the intuition that holistic encoding allows the model to resolve coreferences and bridge inferential gaps without incurring retrieval fragmentation. As corpus size grows toward the context limit, the accuracy gap narrows, consistent with the long-context attention degradation documented in RULER (Hsieh et al., 2024).

5.3 Latency Results

CAG eliminates retrieval latency entirely: the online inference path consists solely of cache deserialization, query tokenization, and autoregressive decoding. On HotPotQA, the median response time for CAG is approximately 40% lower than dense RAG and 15% lower than sparse RAG (Chan et al., 2025). The offline preloading cost is a one-time expense; for a typical 50K-token knowledge base and a Llama 3.1 8B model on a single A100 GPU, preloading completes in under three minutes.

6. Design Guidelines and Trade-Off Analysis

The choice between CAG and RAG is governed by three principal factors:

- CAG is well-suited when the domain knowledge base can be expressed in fewer tokens than the model's effective context length. Empirically, this corresponds to sources such as product documentation, legal contracts, enterprise FAQs, or field-specific corpora of bounded scope. When the corpus spans millions of documents, RAG remains the only tractable option.

- Because CAG preloads knowledge offline, any update to the knowledge base requires recomputing the KV cache. For slowly evolving corpora (e.g., regulatory documents, scientific literature in a narrow subdomain), this overhead is acceptable. For high-frequency or real-time data streams, RAG's on-the-fly retrieval is more appropriate.
- When queries are semantically diverse and the relevant passages constitute a small fraction of a large corpus, RAG's targeted retrieval can be more token-efficient than injecting the entire knowledge base. Conversely, for question sets that require cross-document reasoning "the hallmark of multi-hop tasks" CAG's unified context enables reasoning patterns that retrieval pipelines structurally cannot replicate.

A hybrid CAG-RAG architecture naturally emerges from this analysis: high-level summaries of large corpora are preloaded via CAG, while detailed passages are fetched on demand by a lightweight retrieval module. This stratified approach captures the latency and reliability benefits of CAG for summary-level reasoning while extending coverage to arbitrarily large knowledge stores.

7. Limitations and Future Directions

Despite its advantages, CAG faces several open challenges that motivate ongoing research. As demonstrated by RULER (Hsieh et al., 2024), LLMs exhibit measurable degradation in their ability to exploit information at arbitrary positions within long contexts. Addressing this limitation through improved positional encodings, sparse attention patterns, or hierarchical caching represents a critical direction for making CAG viable over larger corpora. The offline preloading model is unsuitable for knowledge bases subject to continuous updates. Future work may explore incremental KV cache updates, where only the delta between successive corpus versions is re-encoded, rather than recomputing the entire cache from scratch.

KV caches for long contexts consume substantial GPU HBM. Techniques such as quantized caching, sparse attention-head eviction, and cross-layer sharing offer promising avenues for reducing the memory footprint without proportional accuracy loss. The existing empirical record for CAG is largely confined to extractive QA benchmarks. Evaluating CAG on generation-heavy tasks (summarization, dialogue, code generation with private APIs) and on corpora spanning 500K–1M tokens will be necessary to characterize its full operating envelope.

8. Conclusion

This research article shared novel approach to use Context-Augmented Generation (CAG). This is a thoughtful solution to the problems found in traditional retrieval-based systems. It takes advantage of the larger context windows in today's advanced language models and the efficiency of precomputed key-value caches. This makes CAG faster, simpler, and often more accurate than Retrieval-Augmented Generation (RAG), especially when dealing with knowledge bases that are limited in size and stable. CAG doesn't completely replace RAG. Instead, it changes the line between when you need to retrieve information and when you don't. It gives professionals a new, high-performance option for situations where the knowledge base is manageable and rarely change. As language models continue to support even larger context windows and the management of key-value caches improves, CAG will become more widely useful. The presented research will make it a key part of how language models are used in real-world applications.

References

Bhatt, S., & Mohammad, A.F. (2025). AI – Next RAG Frontier: Self-Improving Retrieval v Unifying Explicit and Implicit User Signals. *RAIS Journal for Social Sciences*, 9(2), 495-500.

Bhowmick, A., Mohammad, A.F., Kalva, S. (2026). Agentic AI & LLM Incorporation with Personas. In: H. R. Arabnia, L. Deligiannidis, S. Amirian, F. Ghareh Mohammadi, & F. Shenavarmasouleh (Eds.), *AI Revolution: Research, Ethics and Society (*AIR-RES 2025. Communications in Computer and Information Science, vol 2721). Springer.

Chan, B. J., Chen, C. T., Cheng, J. H., & Huang, H. H. (2025, May). Don't Do RAG: When Cache-Augmented Generation is All You Need for Knowledge Tasks. In *Companion Proceedings of the ACM Web Conference 2025 (WWW Companion '25),* https://doi.org/10.1145/3701716.3715490

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., ... & Wang, H. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv preprint arXiv:2312.10997*.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., ... & Vasic, P. (2024). The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783.*

Hsieh, C. P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., ... & Ginsburg, B. (2024). RULER: What's the Real Context Size of Your Long-Context Language Models? In *Proceedings of the First Conference on Language Modeling (COLM)*.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems (NeurIPS), 33,* 9459–9474.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., ... & Dean, J. (2023). Efficiently Scaling Transformer Inference. In *Proceedings of Machine Learning and Systems (MLSys), 5*, 606-624.

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016, November). SQuAD: 100,000+ Questions for Machine Comprehension of Text. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 2383–2392).