Cache-Augmented Generation (CAG) and Retrieval-Augmented Generation (RAG) Model Enhanced with Cross-Encoder Reranker for Northeastern's Office of Global Studies

David Johnson
Himanshu Tahelyani

Northeastern University

DS 5500 Capstone: Application in Data Science

Spring 2025

March 11, 2025

Abstract

Navigating complex university websites for critical information is often frustrating, especially for international students seeking guidance on visas, applications, and work opportunities. To address this challenge, we developed an Augmented Generation system that combines Retrieval-Augmented Generation (RAG) and Cache-Augmented Generation (CAG) to provide accurate, efficient, and context-aware responses.

The system utilizes web-scraped content from Northeastern University's Office of Global Services (OGS) website, structuring it into a retrievable knowledge base. RAG dynamically retrieves relevant documents using Snowflake/snowflake-arctic-embed-l-v2.0 embeddings and a Cross-Encoder reranker to improve response accuracy. CAG precomputes key-value (KV) caches for frequent queries, significantly reducing response time.

Our evaluation tested different retrieval sizes (5, 10, and 20 documents per query) and applied multiple automated scoring methods—including Token-Level F1, Embedding Similarity, Zero-Shot NLI, and Text Classification relevance checks—to assess response accuracy. While performance was consistent across different retrieval sizes (70

The final system integrates efficient LLMs such as LLaMA-3.2 1B Instruct (RAG) and Phi-3.5-mini-instruct (CAG), balancing performance, accuracy, and resource constraints. This approach ensures both high-quality responses for complex queries and low-latency performance for common requests, making the system scalable and adaptable for real-world deployment.

Contents

Chapter 1

Introduction

1.1 Background

Northeastern University's Office of Global Studies (OGS) provides critical guidance for international students. Given the complexity and volume of queries related to study abroad, visas, and policies, efficient and accurate information dissemination is paramount. An Augmented Generation approach leveraging Retrieval Augmented Generation (RAG) with advanced embedding models as well as Cache Augmented Generation (CAG) with modern large-context LLMs and transformer-based architectures promises to significantly improve service quality and operational efficiency

1.2 Problem Statement

Navigating complex websites to find critical information can be challenging, especially when users are unfamiliar with the layout or terminology. This issue is particularly significant for international students seeking vital information about visas, applications, job opportunities, co-ops, and travel requirements from the Northeastern's Office of Global Services website (hereinafter referred to as *the website*). The current structure of the site makes it difficult to quickly locate specific information, leading to frustration and delays.

This project addresses the problem by creating an Augmented Generation LM application that allows users to query information in natural language and receive accurate, contextually relevant responses with references to the source content. By integrating automated scraping and updates, the system will ensure that the information remains current without requiring manual intervention.

Our team selected this problem because it directly impacts the student experience and offers an opportunity to build practical skills in natural language processing (NLP), web scraping, and cloud-based deployment of AI models.

1.3 Objectives

The objectives of this project are:

- Develop a sophisticated augmented generation system that combines Retrieval-Augmented Generation (RAG) and Cache-Augmented Generation (CAG) to improve response accuracy and efficiency.
- Enhance information relevancy and accuracy using Cross-Encoder reranking to refine document selection in the RAG pipeline.
- Optimize inference speed by leveraging CAG, enabling faster responses for frequent queries by precomputing and caching key-value (KV) pairs.
- Utilize efficient LLMs, balancing performance, accuracy, and resource constraints, ensuring scalability for real-world use.

1.4 Scope

The scope of this project includes:

- Develop a sophisticated augmented generation system that combines Retrieval-Augmented Generation (RAG) and Cache-Augmented Generation (CAG) to improve response accuracy and efficiency.
- Dataset creation through web scraping of the Northeastern OGS website, structuring content into retrievable knowledge segments.
- Embedding model selection to improve retrieval quality, transitioning from sentence-transformers/all-MiniLM-L6-v2 to Snowflake/snowflake-arctic-embed-l-v2.0 for better semantic representation.
- Reranking strategy optimization, incorporating Cross-Encoders to prioritize the most relevant retrieved documents.
- CAG implementation, precomputing KV caches for frequent queries to reduce latency and computational overhead.
- Architecture design, integrating RAG for dynamic retrieval and CAG for high-speed, precomputed responses.
- Systematic evaluation, testing response quality, retrieval accuracy, and computational efficiency across different query scenarios.

Chapter 2

Methodology

2.1 Data Sources

We developed our dataset by scraping more than 300 webpages directly from Northeastern University's OGS website using Scrapy and BeautifulSoup. These pages contained detailed visa policies, procedural instructions, FAQs, and other student-focused content. Each page was parsed into structured segments representing distinct knowledge points, resulting in a dataset of around 973 documententries.

2.2 Data Description

The dataset currently includes 319 pages, with an average of 7,061 characters per page. Each page consists of multiple knowledge segments, but they are unordered and untagged, requiring structured parsing. The primary challenge is handling duplicate or overlapping content, as some pages summarize information found elsewhere. While formatting is mostly consistent, header and section tags may be leveraged for document segmentation.

A key limitation is that only Northeastern's internal guidance is included, meaning broader immigration or visa-related queries may lack sufficient context. Future improvements may involve expanding the dataset with official government sources.

2.3 Data Preparation

Data preprocessing included normalization, removal of stopwords, and text cleaning. Initially, embeddings were generated using sentence-transformers/all-MiniLM-L6-v2, but further analysis revealed semantic information loss due to limited dimensionality (364 dimensions). Consequently, extensive research into embedding models, leveraging Hugging Face leaderboards and benchmarks, led

us to adopt Snowflake/snowflake-arctic-embed-l-v2.0 (1024 dimensions), significantly improving semantic detail representation.

2.4 Embedding Model Comparison

Parameter	**MiniLM-L6-v2**	**Snowflake-Arctic-Embed-l-v2.0**
Embedding Dimension	364	1024
Model Size	22.7M parameters	568M parameters
Semantic Representation	Moderate	High (captures nuanced details)
Training Data	1B+ sentence pairs	Extensive, diverse corpus

Snowflake/snowflake-arctic-embed-l-v2.0 takes approximately 2x the processing time compared to **sentence-transformers/all-MiniLM-L6-v2** with an average retrieval generation time of 0.032s compared to 0.017s. This increase in computation time is attributed to its larger embedding dimensionality (1024 vs. 364) and greater model complexity, which allows for superior semantic representation but comes at the cost of higher latency.

Chapter 3

Selection of Models

3.1 RAG: Model Selection and Resource Considerations

After rigorous comparisons using Hugging Face leaderboards, we strategically selected the LLaMA-3.2 1B Instruct model, primarily due to its superior ability to precisely adhere to user instructions and its considerably lower computational demands compared to larger (e.g., 3B, 7B parameter) models. This selection helped ensure rapid inference, low latency, and resource efficiency.

In addition, all model downloads and deployments were facilitated via Hugging Face, streamlining access to pre-trained models and significantly accelerating development.

3.2 CAG: Model Selection and Resource Considerations

- meta-llama/Llama-3.1-8B-Instruct is an 8-billion-parameter model tuned for instruction following. Its strong conversational abilities and aptitude for handling multi-step instructions make it ideal for cache-augmented generation tasks that demand clear, context-aware responses. However, its large size requires more GPU memory and compute, often necessitating aggressive quantization on systems with limited VRAM. On the plus side, it supports a 128k context length, which is a significant advantage when working with extended contexts.

- microsoft/Phi-3.5-mini-instruct is a lighter, efficient alternative with roughly 3.82 billion parameters and fine-tuning for instruction following. Its compact size makes it a natural fit for hardware with limited memory, and it excels at staying on task and using the provided context—critical for CAG systems that must closely follow cached documents. Although its smaller

capacity might limit nuance or performance on very complex queries compared to larger models, it strikes a solid balance between efficiency and effectiveness, and it also supports a generous 128k context length.

- mosaicml/mpt-7b-8k-instruct is a 7-billion-parameter model designed for instruction-based tasks that offers an 8k context window, notably smaller than the 128k window available with Phi and Llama. It balances capacity and efficiency, and its extended context window (compared to standard 4k models) enables more comprehensive multi-document context inclusion for extended dialogue and complex synthesis. However, despite having fewer parameters than Llama-3.1-8B-Instruct, its extended context window and tuning may render it more resource-intensive than Phi-3.5-mini-instruct, requiring careful optimization or aggressive quantization to run efficiently on limited hardware.

Final CAG Model Selection: For the CAG implementation we selected microsoft/Phi-3.5-mini-instruct because it offers a combination of efficiency and capability that fits our hardware constraints and application needs. With roughly 3.82 billion parameters, this model is compact enough to run efficiently on the available an 8GB RTX 3070, especially when using quantization techniques to further reduce memory usage and speed up responses. Its instruction tuning ensures that it follows prompts accurately, generating context-aware responses essential for cache-augmented generation.

Another key factor is its support for a large 128k token context window. This extended window allows the model to incorporate extensive context from multiple documents, a critical requirement for our multi-document synthesis tasks. The combination of efficient resource usage, strong instruction-following performance, and a huge context window makes microsoft/Phi-3.5-mini-instruct an ideal choice for our project

Chapter 4

Model Development

4.1 RAG Architecture and Configuration

For this implementation, we're using Phi-3.5-mini-instruct, a small yet powerful dense decoder-only transformer-based model developed by Microsoft. This model is a causal language model (LLM), meaning it predicts the next token based on prior context using an autoregressive transformer architecture.

Key features of the model architecture:

- FAISS-based retrieval to quickly fetch relevant documents.
- A Cross-Encoder reranker, fine-tuned on transformer architectures (BERT/RoBERTa), to improve document relevancy ranking.
- The LLaMA-3.2 1B instruct-tuned model, ensuring detailed and accurate responses through extended contextual understanding and instruction-following capability.

4.2 CAG Architecture and Configuration

The model architecture incorporates:

- Attention Mechanism: The core of the model relies on self-attention, which allows it to weigh different tokens in the input when generating responses.
- Transformer Layers: While the exact number of layers isn't explicitly mentioned in this implementation, models like Phi-3.5 typically have a deep stack of transformer blocks.
- Quantization for Efficiency: To speed up inference and reduce memory usage, we're using 4-bit quantization via BitsAndBytesConfig. This helps optimize the model's footprint while maintaining reasonable accuracy.

4.3 CAG Training

Since this implementation does not train the model from scratch, we skip the usual dataset preparation and training pipeline. Instead, we leverage Phi-3.5 as a pre-trained model, loading it directly from Hugging Face. However, we do prepare a KV cache that acts as a form of "runtime optimization." Instead of fine-tuning the model, we precompute key-value pairs from a structured system prompt, embedding relevant knowledge upfront. This essentially mimics fine-tuning at the cache level without modifying the underlying model weights

4.4 CAG Tuning

This setup doesn't explicitly involve hyperparameter tuning in the traditional sense. However, a few key parameters affect performance:

- Max Tokens in Generation: The function `generate_response` limits the response length to 200 tokens by default (customization allowed), balancing efficiency with completeness.
- KV Cache Trimming: `trim_kv_cache` ensures that the cache doesn't increase after each query, keeping only the relevant portion of the sequence.
- Quantization Configuration: The BitsAndBytesConfig setup tweaks aspects like double quantization and compute data types (e.g., using torch.bfloat16).

Since this approach focuses on inference-time optimization rather than model fine-tuning, most hyperparameter tuning would revolve around adjusting cache size, decoding strategies, and memory constraints.

Chapter 5

Evaluation and Comparison

To gauge how well the model handled queries based on cached knowledge, we ran tests using different sample sizes—5, 10, and 20 documents at a time, testing each doc-count 10x (so 30 total tests). For each n=number of documents we tested n questions, one question generated from each document, and evaluated the generated answer against the true answer that should have been produced given that individual documents context. The final average accuracy scores for each group were:

- 5-document tests: 72
- 10-document tests: 68
- 20-document tests: 70.5

Interestingly, while performance varied slightly across different sample sizes, there wasn't a clear trend indicating that more documents consistently improved accuracy.

5.1 Key Evaluation Metrics

To evaluate how well the model-generated responses aligned with the expected answers, we used several different techniques:

- Token-Level F1 Score – A simple word overlap measure that checked if the response contained key terms from the expected answer.
- Embedding Similarity – Used sentence-transformer embeddings to compare the overall semantic meaning.
- Zero-Shot NLI (Natural Language Inference) – Assessed whether the generated response logically entailed the expected answer.
- Text Classification as a Relevance Proxy – Checked whether a pre-trained classifier considered the generated response contextually relevant.

For a response to be marked as correct, at least two of these four metrics needed to pass a 0.7 threshold when comparing the generated answer to the true answer. If they didn't, we attempted a fallback check against the full document context to see if the generated answer could still be justified within the context of all combined documents.

5.2 Issues with Full-Context Validation

The fallback full-document check turned out to be too forgiving. It always flipped a failure to a pass, which meant that every response ended up "passing" when checked against the entire document set, even when the answer wasn't actually present. This made it ineffective as a true accuracy check.

For future testing, a stricter approach would be to validate the generated response against each document individually. If a different document (not the one the original question was sourced from) contained the correct information, only then should a failure be flipped to a pass. This would prevent artificially inflated accuracy scores due to overly broad comparisons.

5.3 Final Thoughts

The results show that while the model performs well when leveraging cached knowledge, accuracy isn't perfect and depends on the evaluation method used. The full-context fallback approach needs refinement, and future iterations should focus on more precise validation techniques to ensure correctness rather than inflating scores.

Chapter 6

Discussion and Future Work

By selecting instruct-tuned, resource-efficient models, our system significantly enhanced response accuracy, contextual relevance, and operational efficiency. Future improvements include integrating external authoritative sources to address potential gaps in the Northeastern dataset, optimizing reranking strategies, and further refining embedding methodologies.

Chapter 7

Conclusion

The developed RAG model augmented by a Cross-Encoder reranker and a carefully selected instruct-based LLM successfully addressed the operational challenges faced by the Northeastern OGS. By rigorously researching and carefully selecting appropriate models, we achieved superior performance with minimized computational resources, providing a scalable and efficient template for institutional applications.

Bibliography

[1] Hugging Face. *Transformers and Model Hub*. https://huggingface.co

[2] Scrapy. *Scrapy Web Crawling Framework*. https://scrapy.org

[3] BeautifulSoup. *BeautifulSoup Documentation*. https://www.crummy.com/software/BeautifulSoup/bs4/doc/