Improving Data Compression with Retrieval-Augmented Generation

Faiza Reedi, Joanna Moon,
Raiyan Jamil, Nivid Rakeshbhai Patel

Fall 2025

1 Introduction

Our research aims to investigate how the compression ratios of lossless data compressors can be optimized. Previous work in this field has shown that pre-trained large language models (LLM) can perform as effective general-purpose data compressors [2].

However, compression ratios can be further improved. We intend to apply the framework of Retrieval-Augmented Generation (RAG) to enhance the intelligence of the LLM, thereby improving its prediction capabilities and consequently its efficiency as a data compressor.

2 Research Context and Problem Statement

Efficient lossless data compression remains a fundamental challenge in information theory and computing. As global data generation continues to grow exponentially—with estimates suggesting over 120 zettabytes of data created annually by 2023—optimizing compression algorithms directly reduces storage and transmission costs. Traditional compressors such as Huffman coding, arithmetic coding, and Lempel-Ziv methods rely on manually designed statistical models. While these approaches are efficient for specific data distributions, they lack adaptability and fail to exploit higher-level semantic patterns.

Recent work has demonstrated that large language models (LLMs) can serve as general-purpose compressors by predicting the next token in a sequence and encoding the prediction error using entropy coding techniques [2, 1]. However, existing LLM-based compressors do not use retrieval or external memory mechanisms. This limitation means they are constrained by their finite context windows and inability to recall relevant external information, leading to suboptimal compression ratios for complex or long-form data.

Our research addresses this gap by investigating the use of Retrieval-Augmented Generation (RAG) [3] to enhance LLM-based compression. Specifically, we aim to integrate a retrieval mechanism that dynamically fetches semantically relevant data chunks during compression and decompression. We hypothesize that RAG will improve the predictive accuracy of the LLM, thereby improving overall compression efficiency.

3 Proposed Solution

To address the limitations of existing LLM-based compressors, we propose integrating Retrieval-Augmented Generation (RAG) into the compression pipeline. Our approach leverages RAG's ability to dynamically retrieve semantically relevant data patterns during compression, thereby enhancing the LLM's predictive accuracy and improving overall compression efficiency. The core insight is that by providing the LLM with contextually relevant examples retrieved from a knowledge base, we can significantly improve its next-token prediction capabilities, which directly translates to better compression ratios through arithmetic coding.

3.1 RAG-Enhanced Compression Architecture

Our proposed system consists of three main components: a retrieval module, an augmented LLM predictor, and an arithmetic coder. The retrieval module maintains a vectorized knowledge base of reference documents, encoded using a dense embedding model. During compression, for each document to be compressed, the retrieval module queries this knowledge base to identify the most semantically similar documents based on cosine similarity in the embedding space.

Specifically, given a document D to be compressed, we compute its embedding $\mathbf{e}_D = f_{embed}(D)$ using a pre-trained embedding model (such as Qwen3-Embedding). We then retrieve the top-K most similar documents from our knowledge base by computing:

$$Retrieved(D) = Top - K\left(\{\mathbf{e}_j | \mathbf{e}_j \in \mathcal{KB}, \cos(\mathbf{e}_D, \mathbf{e}_j)\}\right)$$

where $\mathcal{KB}$ represents our vectorized knowledge base of reference documents (e.g., Wikipedia articles), and $\cos(\cdot, \cdot)$ denotes cosine similarity.

3.2 Knowledge Base Construction and Indexing

The knowledge base is constructed by processing a large corpus of reference documents from the target domain. For text compression, we use Wikipedia articles as our reference corpus. Each document is embedded in its entirety using a dense embedding model (Qwen3-Embedding-0.6B), which captures the semantic content of the document. These embeddings are then stored in a FAISS vector database using efficient indexing structures for fast similarity search during compression.

Unlike traditional approaches that chunk documents into smaller pieces, our approach treats each document as a single unit. This allows the retrieval system to identify documents that are semantically related to the entire document being compressed, rather than just local chunks. The knowledge base can be pre-built and persisted to disk, allowing for efficient reuse across multiple compression sessions without rebuilding the index.

3.3 Context Augmentation and Compression Pipeline

Once relevant documents are retrieved, they are concatenated to form a context prefix that precedes the document to be compressed. The key insight is that by conditioning the LLM on semantically similar documents, we improve its ability to predict tokens in the target document.

The complete compression pipeline operates as follows:

1. **Document Retrieval**: For a given document D to be compressed, we retrieve $K = 3$ most similar documents from the knowledge base: $\{R_1, R_2, R_3\} = Retrieved(D)$.

2. **Context Concatenation**: The retrieved documents are concatenated to form a context prefix: $Context = R_1 \oplus R_2 \oplus R_3$, where $\oplus$ denotes concatenation.

3. **Tokenization**: Both the context and the target document are tokenized using the LLM's tokenizer:

   - Context tokens: $C = tokenize(Context)$ with length $|C| = prefix\_length$
   - Document tokens: $D_{tok} = tokenize(D)$

4. **Full Input Formation**: The context and document tokens are concatenated to form the full input sequence: $Input = C \oplus D_{tok}$, along with appropriate attention masks.

5. **Logit Generation**: The full input is passed through the LLM to generate logits (unnormalized log probabilities) for each token position:

   $$Logits = LLM(Input)[:, : -1, :]$$

   The logits are shifted by one position (removing the last token's logits) to align predictions with targets. These logits are converted to float32 precision for accurate probability computation.

6. **Arithmetic Coding**: The logits are converted to probability distributions and used with arithmetic coding to compress only the document portion (excluding the context prefix). The compression function takes:

- Full input token IDs (context + document)
- Corresponding logits from the LLM
- Prefix length to indicate where the actual document starts

Following information theory, the expected compressed length in bits for token t_j is $-\log_2 P(t_j|t_{<j}, Context)$, where the probability P is derived from the logits via softmax.

The critical innovation is that the retrieved context provides the LLM with semantically relevant information, improving its conditional probability estimates for tokens in the target document. Since arithmetic coding directly uses these probabilities, better predictions lead to shorter compressed representations.

3.4 Adaptive Retrieval Mechanism (Optional Enhancement)

To further optimize performance, we may implement an adaptive retrieval strategy that selectively triggers retrieval based on document characteristics or compression performance. For instance, we could:

- Monitor the LLM's prediction entropy to determine when retrieval would be most beneficial
- Use document similarity scores to decide whether retrieved context is relevant enough to help
- Implement a threshold-based system that activates retrieval only for documents where the baseline LLM struggles

This adaptive approach could reduce computational overhead by avoiding unnecessary retrievals when the LLM is already producing good predictions. *Note: This is a nice-to-have enhancement if time permits after the core system is validated.*

3.5 Fine-tuning and Optimization (Optional Enhancement)

To maximize the effectiveness of RAG-enhanced compression, we may employ Low-Rank Adaptation (LoRA) to fine-tune the LLM on domain-specific compression tasks. The fine-tuning objective is to minimize the cross-entropy loss between predicted and actual token distributions while the retrieval module remains frozen:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|c_i|}\log P_\theta(t_{ij}|t_{i,<j}, Retrieved(c_i))$$

where θ represents the LoRA parameters, N is the number of training chunks, and t_{ij} denotes the j-th token in chunk c_i. We would use a rank of 64 for LoRA adaptation and apply it to the query, key, value, and output projection matrices of the transformer layers. *Note: This is an optional enhancement to explore if the basic RAG approach shows promising results.*

3.6 Expected Improvements

By integrating RAG into LLM-based compression, we expect several key improvements:

1. **Enhanced Prediction Accuracy**: Retrieved contexts provide the LLM with relevant patterns that improve conditional probability estimates, directly reducing entropy and improving compression ratios.
2. **Better Handling of Long-Range Dependencies**: RAG enables the LLM to access information beyond its finite context window, addressing the limitation identified in our problem statement regarding suboptimal compression for long-form data.
3. **Domain Adaptation**: The knowledge base can be customized for specific data domains (text, images, code), allowing the compressor to leverage domain-specific patterns without having to fully retrain the model.

4. **Scalability**: Unlike approaches that rely solely on increasing model size or context length, RAG provides a complementary mechanism for incorporating external knowledge that scales independently of model parameters.

Our approach synthesizes insights from recent advances in RAG architectures with the fundamental connection between prediction and compression established by arithmetic coding. By improving the LLM's predictive capabilities through dynamic knowledge retrieval, we directly address the core bottleneck in LLM-based compression: the accuracy of next-token probability distributions.

4 Evaluation and Implementation Plan

4.1 Evaluation Plan

Our evaluation strategy will rigorously measure whether RAG-enhanced compression achieves better compression ratios than baseline methods. We will conduct a systematic experimental study comparing our approach against established compression techniques.

4.1.1 Datasets

We will evaluate our method on multiple benchmark datasets to ensure robustness across different data types:

1. **enwik8**: A standard 100MB text compression benchmark derived from Wikipedia (commonly used in compression research)
2. **Cosmopedia-100k**: A diverse dataset of educational text documents for testing generalization
3. **Custom test samples**: Small representative documents from various domains to validate correctness during development

For the knowledge base construction, we will use:

- **Wikipedia dumps**: Large corpus of general knowledge documents (e.g., wiki20231101en) for building the retrieval index
- We will index approximately 1000-5000 documents initially, scaling up if computational resources permit

The knowledge base documents and test documents will be completely separate to ensure fair evaluation and prevent information leakage.

4.1.2 Baseline Comparisons

We will compare our RAG-enhanced LLM compressor against:

1. **Traditional compressors**: gzip, bzip2, LZMA (to establish practical baselines and validate our implementation)
2. **Vanilla LLM compressor**: Standard LLM (Qwen3-0.6B or similar) with arithmetic coding but without retrieval (to isolate RAG's contribution—this is our primary comparison)
3. **Larger LLM**: If computationally feasible, test with a larger model to compare RAG-augmented small model vs. larger baseline model

4.1.3 Evaluation Metrics

We will measure the following key metrics:

1. **Compression Ratio**: $CR = \frac{OriginalSize}{CompressedSize}$ (primary metric)
2. **Bits Per Byte (BPB)**: $BPB = \frac{CompressedSizeinbits}{OriginalSizeinbytes}$ (lower is better)

3. **Compression Time**: Wall-clock time for encoding
4. **Decompression Time**: Wall-clock time for decoding
5. **Perplexity Reduction**: Change in LLM perplexity with vs. without retrieved context

4.1.4 Experimental Procedure

For each dataset and method, we will:

1. **Setup**: Build knowledge base index from Wikipedia documents (one-time setup, persisted to disk)
2. **Encode**: Compress test documents using each method:
   - For RAG method: retrieve top-K documents, concatenate as context, pass through LLM, compress with arithmetic coding
   - For baseline: pass document directly through LLM without context, compress with arithmetic coding
   - For traditional: use standard compression tools (gzip, bzip2)
   - Record: compression ratio, compressed size in bytes, compression time, retrieval time
3. **Verify**: Decompress and verify bit-perfect reconstruction (for lossless validation)
4. **Analyze**: Compute metrics and perform statistical analysis:
   - Calculate mean compression ratios across multiple documents
   - Compute standard deviations and confidence intervals
   - Perform paired t-tests to assess statistical significance
   - Analyze per-document results to understand when RAG helps most
5. **Ablation Studies**: Systematically vary parameters:
   - Number of retrieved documents ($K = 1, 3, 5$)
   - Knowledge base size (1000, 3000, 5000 documents)
   - Embedding model variations (if multiple models available)
   - Test document domains (general text, technical text, etc.)
6. **Qualitative Analysis**:
   - Examine which retrieved documents were selected for successful compressions
   - Analyze cases where RAG significantly helped or hurt compression
   - Visualize compression ratio improvements across different document types

4.1.5 Success Criteria

We will consider our research successful if:

1. **Primary Goal**: RAG-enhanced compression achieves statistically significant improvement (p ¡ 0.05) in compression ratio over vanilla LLM compression on at least 2 of 3 major benchmark datasets
2. **Secondary Goal**: The improvement is at least 5% better compression ratio than vanilla LLM
3. **Validation Goal**: Decompressed data is bit-perfect identical to original (lossless verification)
4. **Stretch Goal**: RAG-enhanced compression outperforms traditional methods like gzip on text-heavy datasets

Even if we do not achieve all goals, we will gain valuable insights into when and why RAG helps (or doesn't help) compression, contributing to understanding of the relationship between retrieval and prediction.

4.1.6 Implementation Details

We will implement our system using the following components:

1. **LLM**: Qwen3-0.6B (or similar efficient model) accessed via Hugging Face Transformers, using float16 precision for efficiency
2. **Embedding Model**: Qwen3-Embedding-0.6B for computing document embeddings
3. **Vector Store**: FAISS for efficient similarity search with cosine similarity metric
4. **Arithmetic Coding**: Custom or library-based arithmetic coding implementation that:
    - Takes input token IDs and corresponding logits from the LLM
    - Computes probability distributions via softmax over the logits
    - Encodes tokens using these probabilities
    - Handles prefix lengths to compress only the target document (not the retrieved context)
    - Tracks padding bits for exact reconstruction
5. **Tokenization**: Model-specific tokenizer (e.g., Qwen3 tokenizer)

The core encoding workflow will be:

1. Build and persist the knowledge base index (one-time setup):
    - Load Wikipedia or other reference corpus
    - Embed each document using embedding model
    - Store embeddings in FAISS index
    - Save index to disk for reuse
2. For each document to compress:
    - Embed the document using the same embedding model
    - Retrieve top-K (K=3) most similar documents from the knowledge base
    - Concatenate retrieved documents as context prefix
    - Tokenize context and target document separately
    - Concatenate tokenized context and document (track prefix length)
    - Pass full sequence through LLM to get logits
    - Apply arithmetic coding to compress target document using these logits
    - Store compressed bytes, padding information, and metadata
3. Decompression (future work):
    - Retrieve the same context documents using stored metadata
    - Reconstruct the same LLM input
    - Use arithmetic decoding with LLM logits to reconstruct original tokens
    - Verify bit-perfect reconstruction

Key Technical Details:

- Logits are extracted from LLM and shifted by one position (removing last token) to align predictions with targets
- Logits are converted to float32 for numerical stability during probability computation
- Attention masks are properly constructed for the concatenated context+document input
- The compression function only compresses tokens after the prefix (i.e., the actual document, not the context)
- Memory management includes garbage collection after processing large datasets

4.2 Timeline

We have divided our work into specific phases from now until the end of the academic year (assumed to be late April 2026). This timeline builds in buffer time for inevitable challenges and ensures adequate time for evaluation and poster preparation.

1. **November - December 2025: Environment Setup and Baseline Implementation**
    - Week 1-2 (Nov 20 - Dec 3): Set up development environment, install dependencies (PyTorch, Transformers, FAISS), download and organize datasets (enwik8, Cosmopedia, Wikipedia for knowledge base)
    - Week 3-4 (Dec 4 - Dec 17): Implement vanilla LLM compression baseline:
        - Set up LLM inference (Qwen3-0.6B or similar)
        - Implement tokenization pipeline
        - Get LLM to output logits for given input
        - Verify logit shapes and probability distributions
    - Week 5 (Dec 18 - Dec 24): Implement or integrate arithmetic coding:
        - Understand arithmetic coding algorithm
        - Connect LLM logits to arithmetic encoder
        - Test on small examples, verify lossless reconstruction

    Deliverable by end of December: Working baseline LLM compressor without RAG, verified on small test samples

2. **January 2026: RAG System Implementation**
    - Week 1 (Dec 25 - Jan 7): Holiday break, light work:
        - Load Wikipedia dataset
        - Test embedding model (Qwen3-Embedding) on sample documents
    - Week 2-3 (Jan 8 - Jan 21): Implement RAG retrieval module:
        - Build knowledge base: embed 1000-5000 Wikipedia documents
        - Store embeddings in FAISS index, test persistence to disk
        - Implement retrieval function: query embedding, get top-K results
        - Verify retrieval returns semantically relevant documents
    - Week 4 (Jan 22 - Jan 28): Integrate RAG into compression pipeline:
        - Concatenate retrieved documents as context prefix
        - Modify tokenization to handle context + target document
        - Track prefix length correctly
        - Pass full input through LLM, extract and process logits
        - Feed to arithmetic coder, ensuring only target document is compressed

    Deliverable by end of January: Complete RAG-enhanced compression system, may have bugs but functional end-to-end

3. **February 2026: Testing, Debugging, and Initial Experiments**
    - Week 1-2 (Jan 29 - Feb 11): Debug and validate:
        - Test compression/decompression on small documents
        - Verify lossless reconstruction (bit-perfect match)
        - Fix edge cases (empty documents, very long documents, special characters)
        - Implement compression metrics tracking (Metric class)
    - Week 3 (Feb 12 - Feb 18): Run baseline experiments:
        - Compress test documents with vanilla LLM (no RAG)
        - Compress same documents with RAG-enhanced system

  - Compare compression ratios, record results
  - Compress with gzip/bzip2 for reference
- Week 4 (Feb 19 - Feb 25): Expand experiments:
  - Test on enwik8 (or subset if too large)
  - Test on Cosmopedia documents
  - Analyze when RAG helps vs. doesn't help
  - Document preliminary findings

Deliverable by end of February: Initial experimental results showing RAG vs. baseline comparison

4. **March 2026: Ablation Studies and Optimization**
   - Week 1-2 (Feb 26 - Mar 11): Conduct ablation studies:
     - Vary K (number of retrieved documents): K=1, 3, 5, 10
     - Vary knowledge base size: 1000, 3000, 5000 documents
     - Test different embedding models if feasible
     - Test with different base LLMs if resources allow
   - Week 3 (Mar 12 - Mar 18): Optional enhancements (if time permits):
     - Implement adaptive retrieval (retrieve only when needed)
     - Experiment with different context concatenation strategies
     - Profile code, optimize performance bottlenecks
   - Week 4 (Mar 19 - Mar 25): Finalize experiments and analysis:
     - Ensure all experiments are reproducible
     - Create tables and plots of results
     - Write up technical observations and insights

   Deliverable by end of March: Complete experimental evaluation with ablations, ready for write-up

5. **April 2026: Report Writing and Poster Preparation**
   - Week 1-2 (Mar 26 - Apr 8): Write final research report:
     - Document methodology in detail
     - Present all experimental results with tables and figures
     - Analyze results: what worked, what didn't, why
     - Discuss limitations and future work
   - Week 3 (Apr 9 - Apr 15): Create research poster:
     - Design poster layout
     - Create visualizations (compression ratio comparisons, ablation study results)
     - Write clear, concise descriptions for poster
     - Get feedback from advisor
   - Week 4-5 (Apr 16 - Apr 30): Finalize and present:
     - Practice poster presentation
     - Make final revisions based on feedback
     - Submit final report
     - Present at poster session

   Deliverable by end of April: Polished final report, professional poster, successful presentation

Buffer and Risk Mitigation: We have intentionally kept the March timeline flexible. If we encounter significant technical challenges in January-February (e.g., implementation bugs, computational resource constraints), we can use March to catch up rather than rushing. The optional enhancements (adaptive retrieval, LoRA) are clearly marked as stretch goals that can be dropped if needed.

Key Milestones:

- End of December: Working baseline compressor
- End of January: Working RAG system
- End of February: First experimental results
- End of March: All experiments complete
- Late April: Final poster and presentation ready

This timeline ensures we have at least 4 weeks (all of April) dedicated to evaluation write-up and poster preparation, which is essential for a polished final product. Research always takes longer than expected, so building in this buffer is critical.

5 Revision History

Based on feedback from the course staff and peers, the following changes were made to the proposal:

1. **Clarified Research Gap:** In Section 2, we explicitly articulated that existing LLM compressors fail to utilize external memory, framing this as the primary problem statement.
2. **Quantified Motivation:** Added specific statistics regarding global data growth (180ZB by 2025) to better justify the need for improved compression.
3. **Scope Refinement:** In Section 3.3, we clearly distinguished between "Essential" core deliverables and "Nice-to-Have" features (like Adaptive Retrieval) to address concerns about the project being overly ambitious.
4. **Technical Simplification:** Per peer feedback, we simplified the dense mathematical notation regarding embedding models and fine-tuning to focus on the system architecture and flow.
5. **Formatting Improvements:** Adjusted document margins for better readability and added a placeholder for a system diagram to aid visual understanding of the architecture.
6. **Success Metrics:** Simplified the success criteria to focus on the comparison against the Vanilla LLM baseline.

References

[1] Kecheng Chen, Pingping Zhang, Hui Liu, Jie Liu, Yibing Liu, Jiaxin Huang, Shiqi Wang, Hong Yan, and Haoliang Li. Large language models for lossless image compression: Next-pixel prediction in language space is all you need. 2024.

[2] Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, et al. Language modeling is compression. *arXiv preprint arXiv:2309.10668*, 2023.

[3] Xinyu Gao Kangxiang Jia Jinliu Pan Yuxi Bi Yi Dai Jiawei Sun Meng Wang Haofen Wang Yunfan Gao, Yun Xiong. Retrieval-augmented generation for large language models: A survey. 2023.