Računarski fakultet

Review on the topic of

Retrieval-Augmented Generation (RAG)

Course: Data mining

Advisor:
prof. Nemanja Ilić

Student:
Vanja Kovinić

Belgrade, January 2025.

Contents

1 Introduction

Recent advances in **Large Language Models (LLMs)** have revolutionized natural language processing, demonstrating remarkable capabilities in tasks ranging from text generation to complex reasoning. Models like **GPT-4** [1] and **LLaMA** [2] have achieved good performance in understanding and generating human-like text. However, these models face significant limitations that impact their practical deployment and reliability in real-world applications.

One of the fundamental challenges with traditional LLMs is their reliance on static knowledge encoded in model parameters during training. This approach presents several critical issues.

- First, the knowledge contained within these models becomes outdated as new information emerges, requiring costly and time-consuming retraining or fine-tuning processes.
- Second, the models are constrained by their parameter count, with even the largest models unable to capture all potentially relevant information.
- Third, fine-tuning these models on new data can lead to forgetting, where the model loses previously acquired knowledge while adapting to new information.

To address these limitations, researchers have developed
Retrieval-Augmented Generation (RAG) [3], a novel architecture that combines the generative capabilities of LLMs with dynamic access to external knowledge sources. RAG represents a paradigm shift in how we approach language models, moving from purely parametric knowledge to a hybrid system that can actively retrieve and incorporate relevant information during inference.

The core innovation of RAG lies in its ability to decompose the knowledge acquisition and generation processes. Instead of relying solely on information encoded in model parameters, RAG systems maintain a separate knowledge base that can be easily updated without modifying the underlying language model. This separation provides several advantages: it allows for real-time knowledge updates, reduces the memory requirements for storing information, and maintains consistent access to accurate, verifiable information.

2 Large Language Models: Background and Limitations

2.1 Understanding Language Models

LLMs are probabilistic systems trained to predict the next token in a sequence based on the provided context. During training, these models process vast amounts of text data, learning to capture patterns, relationships, and factual information within their parameters. The knowledge acquired during training is encoded in billions of parameters - for instance, **GPT-3** [4] contains 175 billion parameters, while models like **LLaMA 2** [5] can range from 7 billion to 70 billion parameters.

The training process involves exposing the model to a diverse corpus of text data, allowing it to learn language patterns and accumulate knowledge about various topics. This knowledge, however, is static and bounded by the training cutoff date, creating what is known as the knowledge cutoff problem. Any information or events occurring after the training period are unknown to the model, regardless of their significance.

2.2 Limitations of Traditional LLMs

2.2.1 The Knowledge Cutoff Problem

The knowledge cutoff represents a fundamental limitation of LLMs. For example, a model trained with data up to 2022 would have no inherent knowledge of significant events or technological developments that occurred in 2023. This limitation becomes increasingly problematic as time passes, making the model's knowledge progressively outdated.

2.2.2 Fine-tuning Challenges

Organizations facing the knowledge cutoff problem might consider fine-tuning as a solution. However, fine-tuning presents several significant challenges:

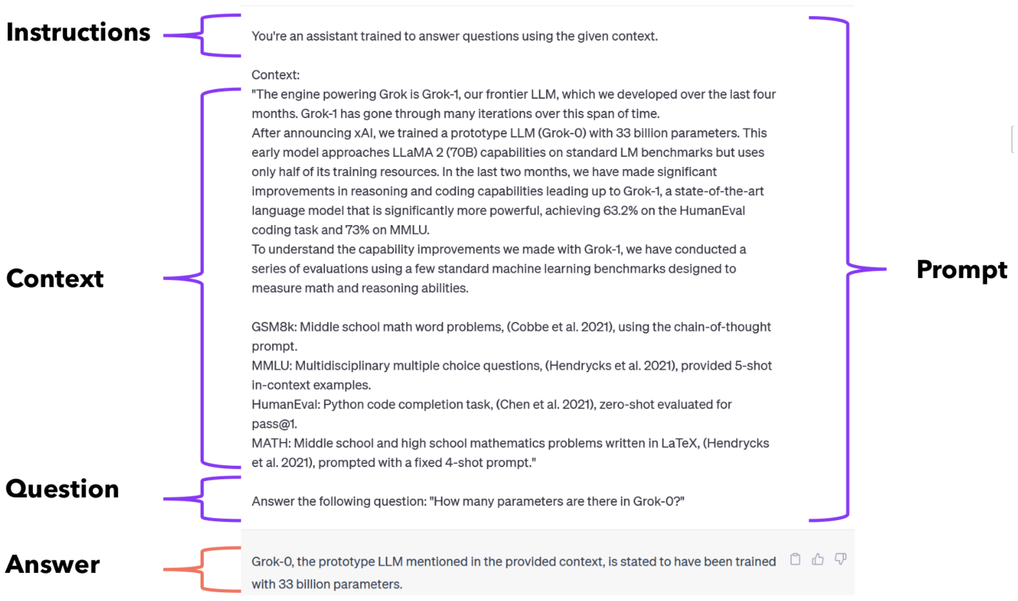
- **Cost Implications:** Fine-tuning large models requires substantial computational resources and expertise, making it prohibitively expensive for many organizations.
- **Parameter Limitations:** The number of parameters in a model may not be sufficient to capture all desired knowledge. Even large models like **LLaMA 2** (70B) have finite capacity for storing information.
- **Forgetting:** Fine-tuning can lead to the model forgetting previously learned information. For instance, a model heavily fine-tuned on recent events might lose its ability to reason about historical contexts.

2.3 Prompt Engineering as a Partial Solution

Prompt engineering has emerged as a technique to mitigate the knowledge cutoff problem by providing relevant context within the input prompt. This approach allows users to "teach" the model new information on the fly, enabling it to reason about and generate responses based on up-to-date information (*Figure 1*).

2.3.1 Benefits of Prompt Engineering

- **Dynamic Knowledge Integration:** Users can provide current information without modifying the model's parameters.
- **Flexibility:** The context can be tailored to specific queries or use cases.
- **Verifiability:** Since the context is explicitly provided, the source of information is transparent and can be verified.



You're an assistant trained to answer questions using the given context.

Context:
"The engine powering Grok is Grok-1, our frontier LLM, which we developed over the last four months. Grok-1 has gone through many iterations over this span of time.
After announcing xAI, we trained a prototype LLM (Grok-0) with 33 billion parameters. This early model approaches LLaMA 2 (70B) capabilities on standard LM benchmarks but uses only half of its training resources. In the last two months, we have made significant improvements in reasoning and coding capabilities leading up to Grok-1, a state-of-the-art language model that is significantly more powerful, achieving 63.2% on the HumanEval coding task and 73% on MMLU.
To understand the capability improvements we made with Grok-1, we have conducted a series of evaluations using a few standard machine learning benchmarks designed to measure math and reasoning abilities.

GSM8k: Middle school math word problems, (Cobbe et al. 2021), using the chain-of-thought prompt.
MMLU: Multidisciplinary multiple choice questions, (Hendrycks et al. 2021), provided 5-shot in-context examples.
HumanEval: Python code completion task, (Chen et al. 2021), zero-shot evaluated for pass@1.
MATH: Middle school and high school mathematics problems written in LaTeX, (Hendrycks et al. 2021), prompted with a fixed 4-shot prompt."

Answer the following question: "How many parameters are there in Grok-0?"

Grok-0, the prototype LLM mentioned in the provided context, is stated to have been trained with 33 billion parameters.

Figure 1: Example where user provides context with answer inside the prompt to enhance the LLMs response [6]

2.3.2 Limitations of Prompt Engineering

However, prompt engineering comes with significant drawbacks:

- **Context Window Constraints:** Every token used for context reduces the available space for the actual query and response. For example, if a model has a 4,096 token limit, providing 3,000 tokens of context leaves only 1,096 tokens for the query and response.
- **Manual Intervention:** Users must manually select and provide relevant context, which can be time-consuming and may require expertise in the subject matter.
- **Context Selection Challenge:** Determining which context is relevant for a given query can be difficult, especially for complex or ambiguous questions.

2.4 The Need for Alternative Approaches

While prompt engineering provides a workable solution for incorporating new knowledge into LLM interactions, its limitations make it suboptimal for many real-world applications. These limitations have motivated the development of more sophisticated approaches to augmenting LLM knowledge, leading to the emergence of Retrieval-Augmented Generation (RAG) as a promising solution. RAG addresses many of the limitations of both traditional LLMs and prompt engineering by providing a systematic way to incorporate external knowledge while maintaining efficiency and scalability.

3 Retrieval-Augmented Generation

3.1 RAG Architecture

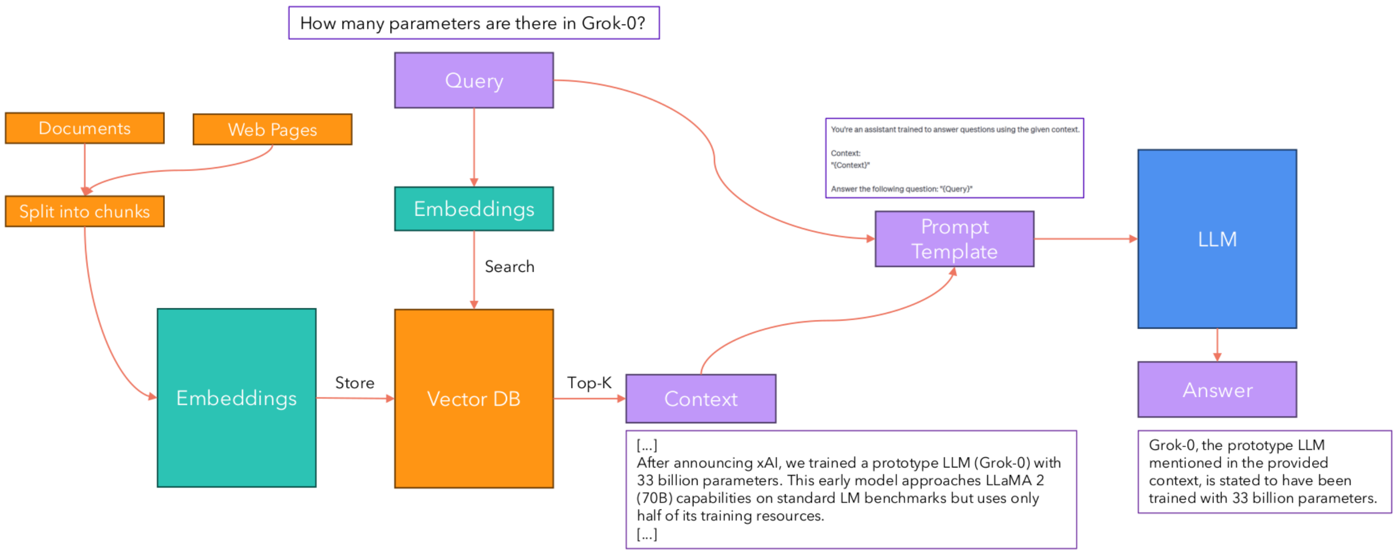
Retrieval-Augmented Generation represents a significant advancement in language model architecture by introducing a dynamic knowledge retrieval component into the generation process. Unlike traditional LLMs that rely solely on their trained parameters, RAG systems maintain a separate knowledge base that can be queried during inference.

3.1.1 Components and Their Interactions

The core components of a RAG system include:

- **Query Encoder:** Transforms user queries into dense vector representations
- **Document Index:** Stores preprocessed documents and their vector embeddings
- **Retrieval System:** Performs efficient similarity search to find relevant documents
- **Generator:** The base language model that produces the final response

These components work in concert, with each stage's output serving as input for the next stage in the pipeline (*Figure 2*).


How many parameters are there in Grok-0?
Query
Documents
Web Pages
Split into chunks
Embeddings
Search
Embeddings
Store
Vector DB
Top-K
Context
You're an assistant trained to answer questions using the given context.
Context:
"{Context}"
Answer the following question: "{Query}"
Prompt Template
LLM
Answer

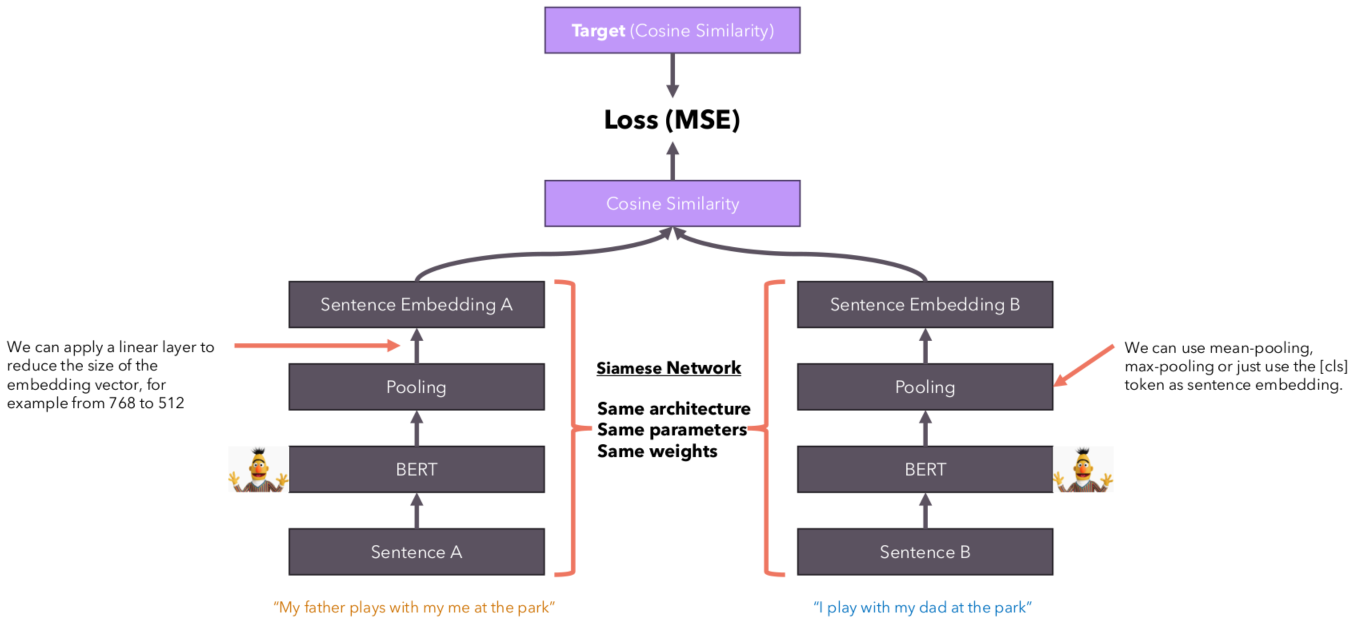
[...]
After announcing xAI, we trained a prototype LLM (Grok-0) with 33 billion parameters. This early model approaches LLaMA 2 (70B) capabilities on standard LM benchmarks but uses only half of its training resources.
[...]

Grok-0, the prototype LLM mentioned in the provided context, is stated to have been trained with 33 billion parameters.

Figure 2: Overview of the RAG pipeline [6]

3.1.2 Query Processing

Query processing begins with the transformation of the user's input query into a dense vector representation. This process, known as query embedding, uses the same encoding mechanism applied to documents in the knowledge base, ensuring compatibility for similarity comparisons. The query encoder typically employs a pre-trained model such as **Sentence-BERT** [**sentence_bert**] to generate these embeddings (*Figure 3*).



Target (Cosine Similarity)
Loss (MSE)
Cosine Similarity
Sentence Embedding A
Sentence Embedding B
We can apply a linear layer to reduce the size of the embedding vector, for example from 768 to 512
Pooling
Pooling
We can use mean-pooling, max-pooling or just use the [cls] token as sentence embedding.
Siamese Network
Same architecture
Same parameters
Same weights
BERT
BERT
Sentence A
Sentence B
"My father plays with my me at the park"
"I play with my dad at the park"

Figure 3: Overview of the RAG pipeline [6]

3.1.3 Document Retrieval

The document retrieval phase involves:

1. Searching the vector database for documents with embeddings similar to the query embedding
2. Ranking retrieved documents based on relevance scores
3. Selecting the top-K most relevant documents to provide as context

3.1.4 Response Generation

The final stage combines the retrieved documents with the original query to generate a response. This process typically involves:

1. Constructing a prompt that incorporates both the query and retrieved context
2. Passing the augmented prompt to the base language model
3. Generating a response that leverages both the model's parametric knowledge and the retrieved information

3.2 Vector Embeddings in RAG

3.2.1 Purpose and Importance

Vector embeddings serve as the foundation of efficient information retrieval in RAG systems. These dense numerical representations capture semantic meaning, allowing for efficient similarity comparisons between queries and documents.

3.2.2 Embedding Generation Techniques

Modern RAG systems typically employ transformer-based models for embedding generation. The process involves:

1. Tokenizing input text into subword units
2. Passing tokens through a neural network architecture
3. Aggregating token representations to create a single vector

3.2.3 Sentence-BERT Architecture

Sentence-BERT (SBERT) represents a significant advancement in embedding generation, specifically optimized for sentence embeddings. Key features include:

- Siamese network architecture for comparing sentence pairs
- Mean pooling of token embeddings (*Figure 4*)
- Training objectives optimized for semantic similarity tasks

The architecture employs a "Siamese" network structure where the same BERT model processes both sentences in a pair, ensuring consistent embedding space for queries and documents.

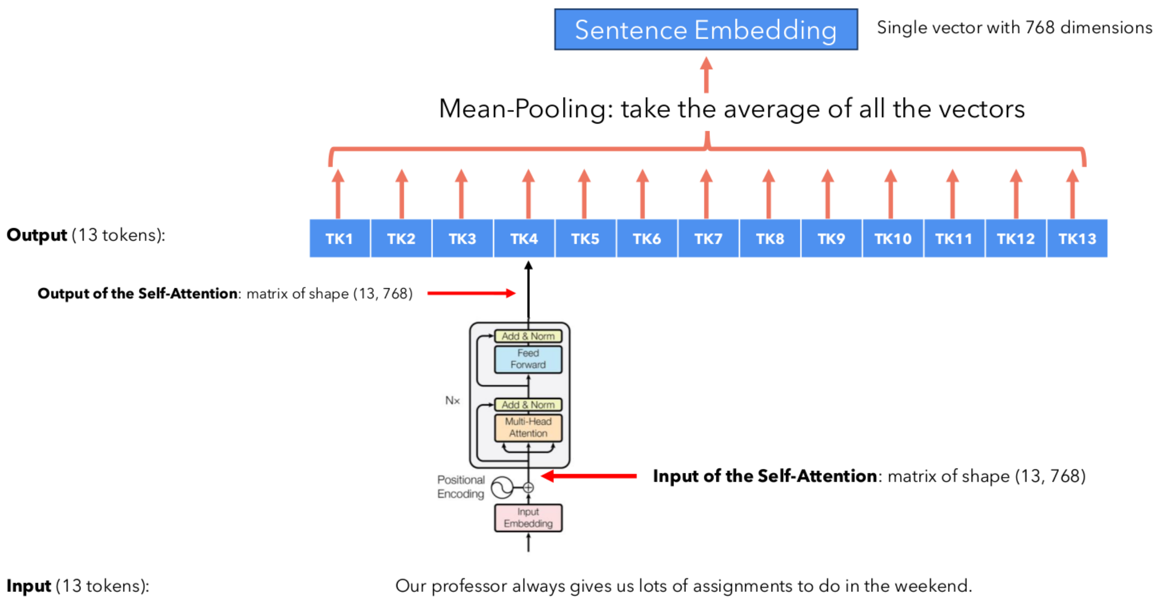

Sentence Embedding
Single vector with 768 dimensions
Mean-Pooling: take the average of all the vectors
Output (13 tokens):
TK1 TK2 TK3 TK4 TK5 TK6 TK7 TK8 TK9 TK10 TK11 TK12 TK13
Output of the Self-Attention: matrix of shape (13, 768)
Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Input of the Self-Attention: matrix of shape (13, 768)
Input (13 tokens):
Our professor always gives us lots of assignments to do in the weekend.

Figure 4: Mean pooling in Sentence BERT architecture [6]

3.2.4 Similarity Metrics

Common similarity metrics used in RAG systems include (*Figure 5*):

- **Cosine Similarity:** Measures the cosine of the angle between vectors, normalized to [-1, 1]
- **Euclidean Distance:** Calculates the straight-line distance between vectors
- **Dot Product:** Computes the scalar product of two vectors

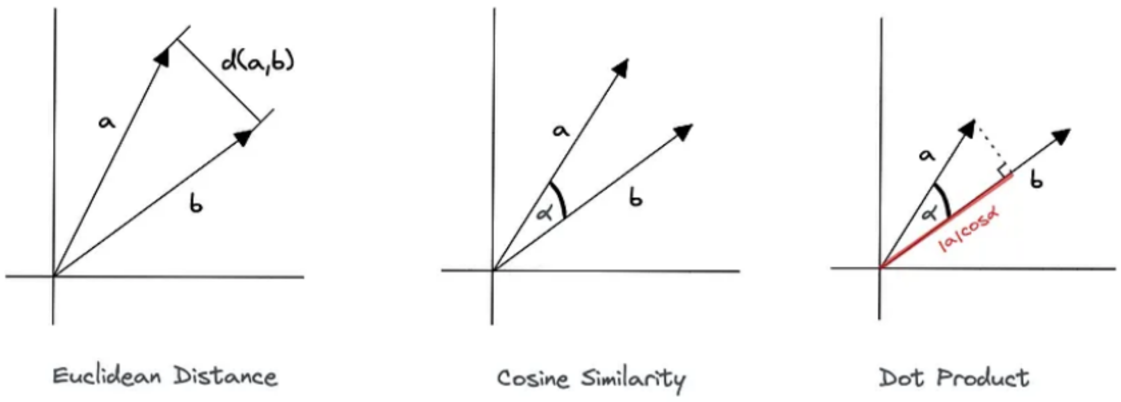


Figure 5: Overview of different similarity metrics used in RAG databases [7]

3.3 Vector Databases

3.3.1 Role in RAG Systems

Vector databases serve as the backbone of efficient information retrieval in RAG systems, providing fast similarity search capabilities and support for real-time updates and queries

3.3.2 Efficient Similarity Search

The challenge of similarity search in high-dimensional spaces is addressed through specialized algorithms that trade perfect accuracy for significant speed improvements. This approach is particularly important in RAG systems where query latency is critical.