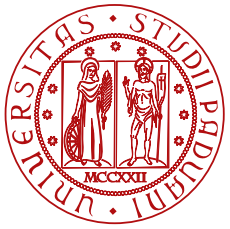

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Department of Information Engineering

Master Thesis in Computer Engineering

Experimental Study on Retrieval-Augmented Generation: Engineering and Evaluation of a Custom RAG system for Open-Domain QA

Master Candidate

Gianluca Antolini

Student ID 2080960

Supervisor

Prof. Nicola Ferro

University of Padova

Academic Year
2024/2025

Date: 8 July 2025

Welcome my son, welcome to the machine.

Abstract

This thesis presents the design, implementation, and evaluation of a custom Retrieval-Augmented Generation (RAG) system for Open-Domain Question Answering (QA). We relied on public open benchmark from the Text REtrieval Conference (TREC) RAG 2024 Track and focused on combining both traditional and innovative Information Retrieval (IR) methods with local Large Language Model (LLM)s to build a flexible and efficient end-to-end pipeline. The retrieval component is based on the Pyserini framework, using datasets and evaluation data and tools provided by TREC (e.g. Microsoft MAchine Reading COmprehension (MS MARCO) Segment v2.1 collection): various retrieval strategies were explored, including Best Matching 25 (BM25), query expansion, Pseudo-Relevance Feedback (PRF), and re-ranking techniques. For the generation component, multiple local LLMs were tested under different prompting strategies and configurations, with particular attention to performance optimization through quantization, Graphics Processing Unit (GPU) acceleration, and fine-tuning. Results were then compared with outputs from State of the Art (SOTA) hosted LLMs to assess relative quality and performance. Additionally, a preliminary experiment with a Parametric RAG (PRAG) approach, a new approach to RAG presented in a recently published paper (January 2025) where context is integrated as model parameters instead of prompt inputs (or both), is introduced. The results highlight how different combinations of retrieval and generation techniques impact the relevance and quality of the final answers: this experimental study contributes to the practical understanding of building customized, efficient, and interpretable RAG systems using open-source tools and local models.

Sommario

Questa tesi presenta la progettazione, l'implementazione e la valutazione di un sistema RAG personalizzato per l'Open-Domain QA. Il lavoro si basa sulle fondamenta del TREC RAG 2024 Track e si concentra sulla combinazione di metodi di IR sia tradizionali che innovativi con LLMs locali, al fine di costruire una pipeline end-to-end flessibile ed efficiente. Il componente di retrieval è basato sul framework Pyserini, utilizzando dataset e strumenti di valutazione forniti da TREC (ad esempio la collection MS MARCO Segment v2.1): sono state esplorate diverse strategie di retrieval, tra cui BM25, query expansion, PRF e tecniche di re-ranking. Per il componente di generazione, sono stati testati molteplici LLMs locali con differenti strategie di prompting e configurazioni, con particolare attenzione all'ottimizzazione delle performance tramite quantizzazione, accelerazione GPU e fine-tuning. I risultati sono stati poi confrontati con gli output di LLMs SOTA online per valutare la qualità e le performance relative. Inoltre, viene introdotto un esperimento preliminare con un approccio PRAG, una nuova metodologia di RAG proposta in un articolo recentemente pubblicato (gennaio 2025), in cui il contesto viene integrato come parametri del modello invece che come input del prompt (o in entrambi i modi). I risultati evidenziano come diverse combinazioni di tecniche di retrieval e generazione influenzino la rilevanza e la qualità delle risposte finali: questo studio sperimentale contribuisce alla comprensione pratica della costruzione di sistemi RAG personalizzati, efficienti e interpretabili utilizzando strumenti open-source e modelli locali.

Contents

List of Figures

List of Tables

List of Code Snippets

List of Acronyms

AG Augmented Generation

AI Artificial Intelligence

Adam Adaptive Moment Estimation

AdamW Adam with Weight Decay

ANN Approximate Nearest Neighbor

ANOVA Analysis of Variance

API Application Programming Interface

BERT Bidirectional Encoder Representations from Transformers

BLEU Bilingual Evaluation Understudy

BM25 Best Matching 25

CNN Convolutional Neural Network

CPU Central Processing Unit

DCG Discounted Cumulative Gain

EM Exact Match

FAISS Facebook AI Similarity Search

FFN Feed-Forward Neural Network

GPT Generative Pre-trained Transformer

GPU Graphics Processing Unit

HNSW Hierarchical Navigable Small World

IR Information Retrieval

JSON JavaScript Object Notation

LLAMA Large Language Model Meta AI

LLM Large Language Model

LoRA Low-Rank Adaptation

MAP Mean Average Precision

METEOR Metric for Evaluation of Translation with Explicit ORdering

MMR Maximal Marginal Relevance

MRR Mean Reciprocal Rank

MS MARCO Microsoft MAchine Reading COmprehension

NDCG Normalized Discounted Cumulative Gain

NLTK Natural Language Toolkit

NN Neural Network

NLP Natural Language Processing

P Precision

PEFT Parameter-Efficient Fine-Tuning

POS Part of Speech

PRAG Parametric RAG

PRF Pseudo-Relevance Feedback

QA Question Answering

QRELS Query Relevance Judgments

R Retrieval

RAG Retrieval-Augmented Generation

RMSProp Root Mean Square Propagation

RNN Recurrent Neural Networks

ROUGE Recall-Oriented Understudy for Gisting Evaluation

Rprec R-Precision

SOTA State of the Art

Seq2Seq Sequence to Sequence

T5 Text-to-Text Transfer Transformer

TF-IDF Term Frequency-Inverse Document Frequency

TREC Text REtrieval Conference

TSV Tab-Separated Values

SGD Stochastic Gradient Descent

1

Introduction

In the current era of generative Artificial Intelligence (AI), the landscape of Artificial Intelligence has been significantly transformed by the advent of Large Language Models (LLMs) and their applications that spread across various domains as these models are characterized by their ability to generate human-like text: this has revolutionized downstream tasks such as natural language understanding, translation, summarization, and Question Answering(QA) that were previously constrained by the limitations of traditional rule-based systems and smaller task-specific models. The introduction of transformer architectures has enabled these models to learn from vast amounts of data by capturing complex patterns and relationships within language. However, despite the impressive capabilites of LLMs, many problems and complications arised with text generation tasks: one of them is the need for accurate and relevant Information Retrieval (IR), especially in Open-Domain QA scenarios. Generated text can often be factually incorrect or lack the necessary context to provide accurate answers, leading to a phenomenon known as "hallucination", where the model generates plausible-sounding but incorrect or nonsensical information: this issue is particularly detectable in domains requiring up-to-date or specialized knowledge, such as legal, medical, and scientific fields and is not only a technical challenge but also an ethical and practical concern, as the dissemination of incorrect information can have serious consequences. To address these challenges, the concept of Retrieval-Augmented Generation (RAG) has emerged as a promising solution: it combines the strengths of IR, a field that has been foundational in computer science for decades, with the generative capabilities of LLMs; with this

approach, models can rely on their own internal knowledge for general understanding while also accessing external, up-to-date information from large corpora or databases. This hybrid approach not only enhances the accuracy and relevance of generated responses but also mitigates the risk of hallucination by grounding the generation process in real-world data. Of course the integration of retrieval and generation components in a seamless manner is crucial for the success of RAG systems and that comes with many challenges in both of the components, in fact, retrieval faces problems such as indexing and searching large datasets and the component in the RAG pipeline must be able to efficiently and effectively identify relevant documents or passages from this data based on the input query, while the generation component must be capable of synthesizing information from these retrieved documents to produce coherent and contextually appropriate responses, creating a need for specialized and fine-tuned LLMs that can handle the nuances of the task at hand. The growing interest in RAG has led to the development of various frameworks and methodologies aiming at the optimization of the retrieval-generation pipeline: this includes advancements in retrieval techniques, such as the use of dense vector representations (instead of sparse ones) and neural retrieval models (which have shown significant improvements over traditional keyword-based methods), while the integration of LLMs with retrieval systems has been explored using various prompting strategies, fine-tuning approaches, and model architectures that improve the generation quality while maintaining efficiency. In this thesis, we aim to explore the State of the Art (SOTA) in this field, focusing on the advancements in both components by testing and evaluating different retrieval strategies and generation techniques in a modular and flexible way, allowing the creation of a customizable RAG system that can be adapted to various Open-Domain QA tasks: by leveraging the latest developments in IR and generative models, we aim to contribute to the understanding of how these components can be effectively combined to improve the performance and reliability of these systems.

This thesis is organized as follows: Chapter 2 presents the technical background useful for understanding the thesis, including fundamentals of IR, LLMs, relevant datasets, evaluation metrics, tools, and an introduction to RAG, as well as a description of the Text REtrieval Conference (TREC) RAG 2024 Track and its evaluation framework; building on this foundation, Chapter 3 details the design and implementation of the developed RAG system, describing its modular ar-

chitecture (retrieval and generation modules), the integration of PRAG concepts , and the evaluation methodology, while Chapter 4 reports on the experimental setup and results, including retrieval and RAG experiments, comparisons with other participants of the Track, and analysis of the final outcomes. Finally, Chapter 5 concludes this document by summarizing the main findings, discussing the contributions of the thesis, and outlining possible directions for future research.

2 Background

2.1 Technical Background

2.1.1 Information Retrieval

IR is the discipline concerned with obtaining relevant information from large collections of unstructured/semi-structured data, typically text documents. Its primary goal is to satisfy a user's information need (expressed as a query) by returning a ranked list of documents/passages that are most likely to be relevant. IR systems are in fact implemented in search engines, digital libraries, and other modern applications that require access to textual information in a scalable and efficient manner. It is not a new field on informatics as its origins can be traced back to the 1950s and 1960s: the first systems were actually designed to help researchers and librarians retrieve scientific literature from digital archives as they were growing substantially and, as the advent of the web took place, over the decades, it evolved from simple keyword matching to sophisticated probabilistic and machine learning-based models. The core compontents of an IR system can be summarized as follows:

- **Document Collection**: Gathering and organizing the corpus of documents to be searched.

- **Indexing**: Processing the collected documents to build data structures (**indexes**) that enable fast retrieval.

- **Query Processing**: Interpreting and transforming user queries into a form suitable for searching.
- **Retrieval**: Matching the query against the index to identify candidate documents to retrieve.
- **Ranking**: Ordering the retrieved documents based on estimated relevance to the query, using retrieval models or scoring functions.
- **Evaluation**: Measuring the effectiveness of the system using metrics such as precision, recall, and Normalized Discounted Cumulative Gain (NDCG).

Indexing and Searching

Indexing is a fundamental part in an IR system: it enables efficient and scalable search over large document collections that can be unstructured and not easily parseable. An **index** is a data structure that maps terms or features extracted from documents to their locations within the collection to allow for rapid lookup and retrieval. The most common type of index in text retrieval is the **inverted index**, where each unique term in the corpus is associated with a list of documents (or positions within the documents) where the term appears: this is a structure that facilitates the system to quickly identify all documents containing a given query term (instead of scanning the whole dataset), thus significantly reducing search time.

Index Construction:

- **Tokenization**: Documents are split into tokens (words or terms).
- **Normalization**: Tokens are standardized (e.g., lowercasing, stemming, removing stopwords).
- **Index Population**: For each token, the index is updated to record the document and position where it occurs.

Types of Indexes:

- **Sparse Indexes:** They represent only the non-zero or present features in the data and are typically used for text retrieval where most terms do not appear in most documents.

  - *Inverted Index*: Maps terms to lists of documents (most common for text). This is the main example of a sparse index, as it only stores entries for terms that actually occur in documents and nothing else.
  - *Positional Index*: Extends the inverted index by storing also term positions to enable phrase and proximity queries.
  - *Forward Index*: Maps documents to lists of terms. This can be considered a sparse index if only present terms are stored per document.

- **Dense Indexes:** Indexes that represent all features for each document or query, typically used in neural IR where every document/query is mapped to a dense vector.

  - *Dense Vector Index*: It stores dense embeddings for documents and/or queries, often indexed with Approximate Nearest Neighbor (ANN) structures (e.g., Hierarchical Navigable Small World (HNSW), Facebook AI Similarity Search (FAISS)). Here, every document is represented by a fixed-size vector, regardless of term presence, thus the size of a dense index can be significantly larger than a sparse index.

Searching: When a user submits a query, the system processes it (applies tokenization, normalization and other standardizations methods) and uses the index to retrieve candidate documents: for keyword-based queries, the **inverted index** is used to find documents containing the query terms, while for neural or dense retrieval, the query is embedded into a vector and compared (e.g., via cosine similarity) to document vectors in the dense index. The main algorithms used for searching can be categorized as follows:

- **Boolean Retrieval**: Returns documents that exactly match the logical conditions of the query.

- **Ranked Retrieval**: Scores and ranks documents based on relevance to the query, using models such as BM25 or neural ranking functions.

- **Hybrid Approaches**: Combine sparse (term-based) and dense (embedding-based) indexes for improved effectiveness.

Retrieval Models

Retrieval models are divided in three main categories based on the index type on which they operate and the way they represent documents and queries. These models are:

- **Sparse models**: also known as lexical or term-based models, they represent documents and queries as high-dimensional sparse vectors, where each dimension corresponds to a unique term in the vocabulary. The most common example is the *Bag-of-Words* model, where the presence or frequency of terms is used for matching. Classic scoring functions include Term Frequency-Inverse Document Frequency (TF-IDF) and BM25 (which we will discuss about later in this section of the chapter). These models are highly efficient, fast, interpretable and require less computational power but they rely on exact term overlap, which limits their ability to capture semantic similarity (e.g., synonyms or paraphrases), creating a vocabulary mismatch problem.

- **Dense models**: these neural retrieval models use Neural Network (NN)s (often based on transformer architectures) to encode documents and queries into low-dimensional dense vectors (embeddings) and the retrieval is performed by measuring the similarity (for example with cosine similarity or dot product) between these embeddings. They can capture semantic relationships beyond exact term matches, thus enabling better generalization and retrieval of relevant documents, even when the lexical overlap is small or not even present. They however require significant computational resources for training and inference and are also less interpretable than sparse ones.

- **Hybrid models**: they offer a balance between the two models above, often achieving the best effectiveness at the cost of increased system complexity.

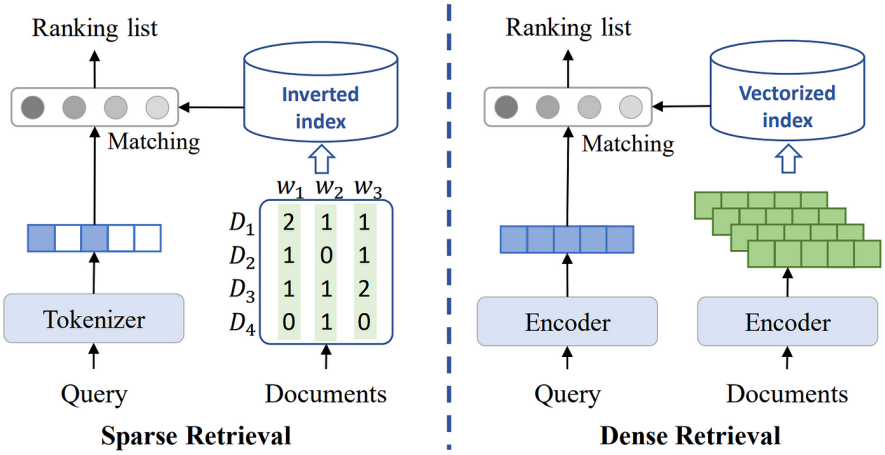

Figure 2.1: Sparse and Dense Retrieval compared

BM25

This model has been developed in the 1990s as part of the Okapi family of retrieval functions (developed at London's City University, [1]) and has been a breaktrough in the field of IR as it introduced a probabilistic approach to document ranking: it is based on the idea that the relevance of a document to a query can be estimated by considering the **TF-IDF** of the terms in the query, along with some additional parameters that account for document length normalization. The BM25 score for a document with respect to a query is computed as follows:

$$\text{BM25}(d, q) = \sum_{t \in q} \text{IDF}(t) \cdot \frac{f(t, d) \cdot (k_1 + 1)}{f(t, d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{\text{avgdl}})} \tag{2.1}$$

where:

- $f(t, d)$ is the term frequency of term t in document d,
- $|d|$ is the length of document d,
- avgdl is the average document length in the collection,
- k_1 and b are hyperparameters (commonly $k_1 \approx 1.2$–2.0, $b \approx 0.75$),
- $\text{IDF}(t)$ is the inverse document frequency of term t.

BM25 improves over simple **TF-IDF** by introducing document length normalization and tunable parameters, making it robust across different collections

and query types, in fact it remains a strong baseline in modern IR systems and is often used as the first-stage retriever in multi-stage pipelines (e.g., in search engines), where its results may be further reranked by more sophisticated (e.g., neural) models.

Retrieval Techniques

Retrieval effectiveness can be further improved by applying various techniques (the combination of multiple ones too) on top of basic retrieval models to improve recall and precision, with the most common approaches including:

Re-ranking: After an initial set of candidate documents is retrieved, a more sophisticated model is used to rerank these candidates for better relevance estimation. SOTA rerankers such as **MonoT5** and **MonoBERT** are the most popular choices since these models can take a query-document pair as input and output a relevance score, making fine-grained ranking based on deep semantic understanding of the text possible.

- **MonoT5**: A reranker based on the Text-to-Text Transfer Transformer (T5) [2] architecture fine-tuned for retrieval tasks that encodes both the query and document into a joint representation, allowing it to capture complex relationships. It is the most commonly used reranker in modern IR systems, as it can be fine-tuned on specific datasets to improve its performance.

- **MonoBERT**: Similar to MonoT5, but based on the Bidirectional Encoder Representations from Transformers (BERT) architecture [3], thus using a bidirectional transformer to understand context and semantics in both queries and documents.

One of the most utilized methods in this particular field is **MMR** [4]: it is an algorithm that select documents/passages and reranks them in order to balance relevance and diversity (it can be controlled by a parameter λ), reducing redundancy in the results by penalizing documents that are too similar to those already selected and thus improving the diversity of information presented to the user. The MMR score for a document d with respect to a query q and a set of already selected documents S is computed as follows:

$$\text{MMR}(d, q, S) = \lambda \cdot \text{Relevance}(d, q) - (1 - \lambda) \cdot \max_{s \in S} \text{Similarity}(d, s) \tag{2.2}$$

where:

- Relevance(d, q) is the relevance score of document d with respect to query q (e.g., BM25 score),
- Similarity(d, s) is a similarity measure (e.g., cosine similarity) between document d and a document s already in the set S.
- λ is a parameter that controls the trade-off between relevance and diversity (typically set between 0 and 1).
- S is the set of already selected documents.

Query Expansion: This technique aims to augment the original user query with additional terms (e.g., synonyms or related concepts) to improve recall and address vocabulary mismatch. The expansion terms can be derived from external resources (like thesauri or word embeddings), from top-ranked documents (e.g., with **PRF**) or, in more modern approaches, from the context of the query itself using LLMs to generate semantically related terms. In **PRF** [5], the system assumes that the top-ranked documents from an initial retrieval are relevant and extracts terms from them to expand the query in an iterative process that can help surface more relevant documents that may not match the original query terms exactly and that does not need any external resources or high computational power.

Tools and Frameworks

A variety of tools and frameworks have been developed to support modern IR research and applications, enabling efficient indexing, retrieval, and evaluation of large-scale text collections. Among the most widely used, we here cite the following ones, which are propedeutic to the work done in this thesis:

- **Anserini** [6]: it is an open-source IR toolkit built on top of the *Apache Lucene* search library, providing a reproducible research platform for traditional and neural retrieval models that supports standard indexing, retrieval pipelines and integration with evaluation tools.
- **Pyserini** [7]: it's the Python interface to Anserini, designed to make IR experimentation more accessible given the widespread use of the cited programming language in AI field. It allows users to easily build, search, and

evaluate indexes using both sparse and dense retrieval models, supporting integration with pre-trained transformer models and providing utilities for hybrid retrieval, re-ranking, and query expansion.

- **FAISS** [8] is a library for efficient similarity search and clustering of dense vectors developed by Facebook AI Research and commonly used for implementing dense retrieval and ANN search at scale.

2.1.2 Large Language Models

LLMs are a class of NN models designed to process and generate human language at scale: they are typically based on deep learning architectures (most notably transformers, which we will explain in detail in the next section) and are trained on massive corpora of text data. They have rapidly advanced the field of Natural Language Processing (NLP), enabling breakthroughs in tasks such as text generation, summarization, translation, question answering, and dialogue systems, with new models being released frequently by research labs and multinational companies (e.g., OpenAI, Google, Meta, etc.) and new techniques, architectures and processing methods being developed to improve their performance and efficiency. The evolution of these models has followed the broader trajectory of NNs in NLP by moving from traditional Recurrent Neural Networks (RNN)s (based on recurrent architectures in which the output of a layer is fed back as input to the same layer inside the NN) and Convolutional Neural Network (CNN)s (which use convolutional layers to capture local patterns in text) to the transformer architecture, which has become the de facto standard for large-scale language modeling by allowing for efficient parallelization and better modeling of long-range dependencies in text. Modern LLMs, such as Generative Pre-trained Transformer (GPT), BERT, and T5 families are pre-trained on large-scale datasets and can be fine-tuned for a wide range of downstream tasks. LLMs are now widely used in both research and industry due to their ability to understand context, generate coherent and contextually relevant text, and adapt to various applications with minimal task-specific supervision: their impact spans search engines, conversational agents, content creation, code generation, and many more other domains.

Transformers

The Transformer architecture, introduced by Vaswani et al. in 2017 [9], has become the foundation of modern LLMs architectures due to its effectiveness in modeling sequential data and its scalability that (unlike previous architectures), relies entirely on *self-attention* mechanisms: they allow the model to weigh the importance of different tokens in the input sequence when generating representations, enabling it to capture long-range dependencies and contextual relationships without the limitations of recurrence or convolution. For example, in a sentence like *"The cat sat on the mat because it was warm"*, the transformer can learn that *"it"* refers to *"the mat"* and not to *"the cat"* by attending to the relevant parts of the input sequence, even if they are far apart.

Components:

- **Input Embedding**: Each token in the input sequence is mapped to a dense vector and, since Transformers lack recurrence, *positional encodings* are added to these embeddings to inject information about the position of each token.

- **Self-Attention Mechanism**: This layer computes a weighted representation of the entire sequence for each token so the model can focus on relevant parts of the input. For a sequence of input vectors $X = [x_1, ..., x_n]$, the attention mechanism computes:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.3}$$

  where Q (queries), K (keys), and V (values) are linear projections of X, and d_k is the dimension of the key vectors.

- **Multi-Head Attention**: Instead of a single attention function, the model uses multiple heads to capture information from different representation subspaces:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.4}$$

  where each head is an independent attention operation.

- **Feed-Forward Network**: Each position is passed through a position-wise feed-forward network (two linear layers with a non-linearity).

- **Residual Connections and Layer Normalization**: Each sub-layer is wrapped with residual connections and followed by layer normalization to stabilize training.

Even though there are many variants of the Transformer architecture, with new components and modifications being proposed frequently, the original one consists of an *encoder* that processes the input and a *decoder* that generates the output, both composed of stacked layers of the components above. Transformers enable parallel processing of sequences (which leads to faster training compared to sequential models), they are highly effective at modeling long-range dependencies in data through self-attention mechanisms and are flexible and scalable, making them suitable for very large models and datasets. However, this architecture has some major drawbacks: the self-attention mechanism has quadratic computational complexity with respect to sequence length, making the processing of very long sequences computationally expensive and they require large amounts of data and computational resources for effective training.

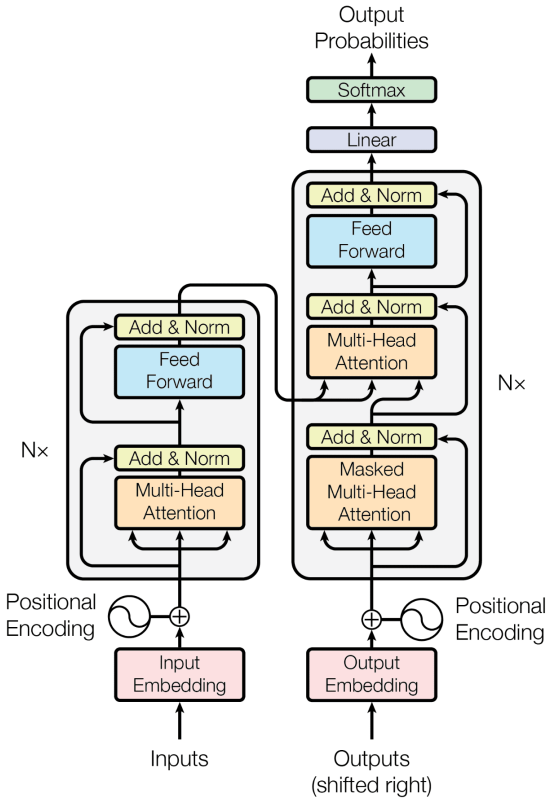

Figure 2.2: Basics Transformer Architecture

Processes

When working with LLMs, several key processes are fundamental to their practical application and effectiveness:

- **Pre-training**: it is the initial phase where a large language model is trained on a massive corpus of text data to learn general language patterns, grammar, facts, and some level of reasoning ability. It is very unlikely that a normal user will have access to the pre-training phase, as it requires significant computational resources and large datasets, thus it is typically performed by large organizations or research institutions. One fundamental process that can be utilized during training is optimization, which is the process of adjusting the model's parameters to minimize a loss function that measures the difference between the model's predictions and the actual data: common optimizers include Stochastic Gradient Descent (SGD), Root Mean Square Propagation (RMSProp), and Adaptive Moment Estimation (Adam) [10]. Being the most widely utilized, the last one combines the advantages of two other extensions of SGD (adaptive learning rate and momentum) and it maintains running averages of both the gradients and their squared values, allowing for efficient and robust training. A relativelty new variant of this is Adam with Weight Decay (AdamW) [11]: it decouples weight decay (L2 regularization) from the gradient update, addressing the issue that standard Adam's weight decay is not applied correctly, leading to better generalization, thus making it the default optimizer for many transformer-based models.

- **Fine-tuning**: it is the process of adapting a pre-trained language model to a specific task or domain by continuing its training on a smaller, task-specific dataset, allowing the model to specialize and improve its performance on particular applications such as QA, summarization, or domain-specific text generation, while leveraging the general language understanding acquired during pre-training. Various techniques can be used for fine-tuning, and they can be broadly categorized into two main approaches:

  - **Full Fine-tuning**: where all model parameters are updated during the fine-tuning process, allowing the model to adapt fully to the new task but requiring significant computational resources and memory.

A common approach of this kind of fine-tuning process is *Linear Projection*, where a linear layer is added on top of the pre-trained model to map the output to the desired task-specific output space (e.g., classification, regression).

– **Parameter-Efficient Fine-Tuning (Parameter-Efficient Fine-Tuning (PEFT))**: where only a small subset of parameters is updated, or additional trainable layers are added to the pre-trained model, thus reducing the computational cost and memory footprint while still achieving good performance on the target task. One of the most popular approaches in this category is LoRA [12]: it is a parameter-efficient fine-tuning technique that, instead of updating all model parameters during the process, injects trainable low-rank matrices into certain layers (typically attention or feed-forward layers) of the pre-trained model. Its core idea is to decompose the weight update ΔW into two smaller matrices A and B such that $\Delta W = AB^\top$, where $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{d \times r}$ with $r \ll d$, thus reducing the number of trainable parameters and memory footprint and making the process feasible even on resource-constrained hardware.

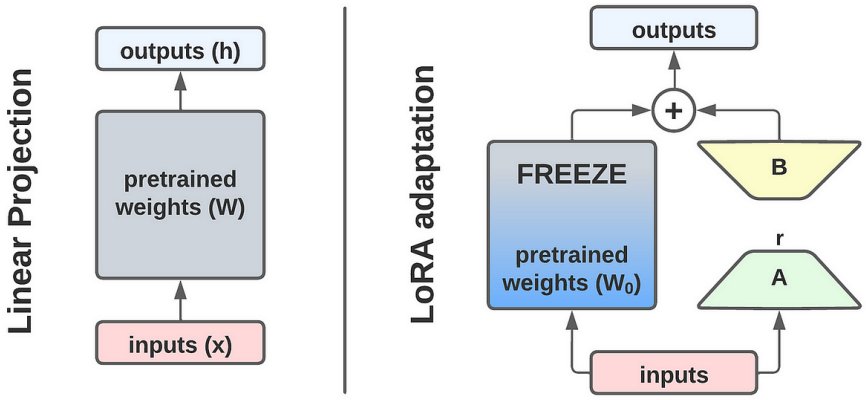

Figure 2.3: Linear Projection compared to LoRA

- **Inference**: this refers to the deployment phase, where the trained or fine-tuned model is used to make predictions or generate outputs based on new, unseen inputs. Typically optimized for speed and efficiency, inference is performed in real-time or at scale in production environments.
- **Prompting**: this process is relatively new as it involves the design of input prompts (the input query or context given to the model) to elicit desired

responses from the model, leveraging its pre-trained knowledge and capabilities. Effective prompting can significantly influence the quality and relevance of the generated outputs, with different architectures and types of LLMs requiring different prompting strategies. For example, when considering the task of summarizing a passage of text, the prompt structure can vary depending on the model. For a chat-based model in the GPT models family, the prompt may include explicit roles:

```
user: Summarize the following text in one sentence:
"Artificial intelligence is a field of computer
science focused on creating systems capable of performing
tasks that typically require human intelligence."
assistant:
```

For an instruction-tuned model (e.g., Meta's Large Language Model Meta AI (LLAMA) models family), the prompt might use special tokens:

```
<s>[INST] Summarize the following text in one sentence:
"Artificial intelligence is a field of computer
science focused on creating systems capable of performing
tasks that typically require human intelligence."
[/INST]
```

For a Sequence to Sequence (Seq2Seq) model like Google's T5, the prompt is typically a plain instruction with a task prefix:

```
summarize: Artificial intelligence is a field of
computer science focused on creating systems capable
of performing tasks that typically require human
intelligence.
```

- **Generation**: it is the process by which LLMs produce human-like text outputs, such as completing a prompt, answering a question, or generating summaries and it can involve various decoding strategies (e.g., greedy search, beam search, sampling) to balance coherence, diversity, relevance and other desired qualities in the generated text.

Tools and Frameworks

Modern LLMs are supported by a rich ecosystem of tools and frameworks that facilitate model development, training, fine-tuning, and deployment:

- **PyTorch** [13]: A widely used deep learning Python framework that offers dynamic computation graphs and intuitive Application Programming Interface (API)s, making it popular for research and production, with many publicly available transformer-based models and LLMs implemented and trained using it.

- **TensorFlow**: Another leading deep learning framework, developed by Google, supporting both research and large-scale production deployments, used for training and serving LLMs, with strong support for distributed training and hardware acceleration.

- **Hugging Face Transformers**: An open-source library that provides thousands of pre-trained transformer models for tasks such as text classification, question answering, summarization, and more, offering a unified API for both PyTorch and TensorFlow, making it easy to load, fine-tune, and deploy SOTA models.

- **PEFT**: A library (developed by Hugging Face) that enables efficient fine-tuning of large models by updating only a small subset of parameters (e.g., via LoRA, adapters).

- **GPU Quantization and Acceleration**: Techniques and libraries such as *bitsandbytes* [14] and *Accelerate* allow for quantizing model weights (e.g., to 8-bit or 4-bit precision), significantly reducing memory usage and speeding up inference and training on GPUs.

2.1.3 Datasets and Evaluation

Datasets and evaluation play a central role in the development and benchmarking of retrieval and generation systems since large, high-quality datasets are essential for training, fine-tuning, and testing models, while robust evaluation metrics and tools ensure that system performance can be measured objectively and compared across different approaches is fundamental.

Datasets

In IR and generation tasks, datasets are used to provide both the source material (documents, passages, or web pages) and the ground-truth labels or answers for evaluation. We here cite two of the most widely used datasets in retrieval and generation tasks, that are propedeutic to the work done in this thesis:

- **MS MARCO v2.1**: this dataset is a large-scale benchmark for passage retrieval and question answering. Its first version was released in 2016 [15], and it has since evolved to include more diverse and complex queries and passages: we here focus on the v2.1 release (an upgrade to the most known v2), which is a significant update that includes both a complete and a segmented version. The following is an example of an entry in the MS MARCO v2.1 dataset (each entry is a JavaScript Object Notation (JSON) object containing a query, a passage, and an answer):


```
{
    "answers": [
        "A corporation is a company or group of people
        authorized to act as a single entity and recognized
        as such in law."
    ],
    "passages": [
        {
            "is_selected": 0,
            "url":
            "http://www.wisegeek.com/what-is-a-corporation.htm",
            "passage_text": "A company is incorporated in a
            specific nation, often within the bounds of a
            smaller subset of that nation, such as a state
            or province. The corporation is then governed
            by the laws of incorporation in that state.
            A corporation may issue stock, either private
            or public, or may be classified as a non-stock
            corporation. If stock is issued, the corporation
            will usually be governed by its shareholders,
            either directly or indirectly."
```

```
            }
            // ...
        ],
        "query": ". what is a corporation?",
        "query_id": 1102432,
        "query_type": "DESCRIPTION",
        "wellFormedAnswers": []
    }
```


- **WebGPT**: it consists of question-answer pairs generated by language models augmented with web search results and includes both queries, retrieved web documents and human-annotated answers, making it suitable for evaluating Open-Domain QA and generation systems that leverage external knowledge. An example of an entry in the WebGPT dataset (it is accessible via HuggingFace interfaces) is as follows:

```
Question

Dataset: triviaqa
ID: 18c654a169eb80287f4353d33e701b1c
Full Text: "Voiced by Harry Shearer, what Simpsons
character was modeled after Ted Koppel?"

Answer 0

Quotes Used:

1. Title: Kent Brockman (en.wikipedia.org)
Extract: "Kent Brockman is a fictional character in
the animated television series The Simpsons. He is
voiced by Harry Shearer and first appeared in the
episode 'Krusty Gets Busted'. He is a grumpy,
self-centered local Springfield news anchor."

2. Title: Krusty the Clown (en.wikipedia.org)
Extract: "Krusty was created by cartoonist Matt
```

```
Groening and partially inspired by Rusty Nails,
a television clown from Groening's hometown of
Portland, Oregon."

Generated Answer:
"The Simpsons character that was possibly based
on Ted Koppel is Kent Brockman. He is a local
news anchor in Springfield and is modeled after
Ted Koppel."

Human Preference Score: 1.0

Answer 1

...
```

Evaluation Metrics

Evaluation metrics are used to quantitatively assess the effectiveness of retrieval and generation models.

- **Retrieval Metrics**:
  - **Per-query metrics:**
    - **NDCG**: Measures ranking quality for a single query by considering the position of relevant documents in the returned list, rewarding highly ranked relevant results. The following is the formula for NDCG for a single query:

$$\text{NDCG@}k = \frac{1}{\text{IDCG@}k} \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{2.5}$$

      where rel_i is the graded relevance of the result at position i, and IDCG@k is the ideal Discounted Cumulative Gain (DCG) (best possible ordering). In practice, NDCG is computed for each query and then averaged across all queries to obtain the final score.

- **Aggregate metrics (across all queries):**
  - **MAP**: Computes the mean of average precision scores for all queries, reflecting both precision and recall across the ranked list. It is calculated as:

$$\text{MAP} = \frac{1}{Q} \sum_{q=1}^{Q} \frac{1}{m_q} \sum_{k=1}^{n} P_q(k) \cdot rel_q(k) \tag{2.6}$$

    where Q is the number of queries, m_q is the number of relevant documents for query q, $P_q(k)$ is the precision at cutoff k, and $rel_q(k)$ is 1 if the item at rank k is relevant, 0 otherwise.
  - **Mean Reciprocal Rank (MRR)**: Focuses on the rank position of the first relevant document, averaging the reciprocal rank over all queries. It is defined as:

$$\text{MRR} = \frac{1}{Q} \sum_{q=1}^{Q} \frac{1}{\text{rank}_q} \tag{2.7}$$

    where rank_q is the rank position of the first relevant document for query q.

- **Generation Metrics**:
  - **F1, Precision, Recall**: Standard metrics for overlap between generated and reference answers, commonly used in extractive QA. The following formulas are used to compute Precision, Recall, and F1:

$$\text{Precision} = \frac{|\text{Prediction} \cap \text{Reference}|}{|\text{Prediction}|} \tag{2.8}$$

$$\text{Recall} = \frac{|\text{Prediction} \cap \text{Reference}|}{|\text{Reference}|} \tag{2.9}$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.10}$$

  - **Exact Match (EM)**: Exact Match measures the percentage of generated answers that exactly match any of the reference answers, often used in QA tasks. The formula for Exact Match is:

$$\text{EM} = \frac{\text{Number of exact matches}}{\text{Total number of examples}} \tag{2.11}$$

– **Bilingual Evaluation Understudy (BLEU)**: Evaluates n-gram overlap between generated and reference texts, widely used in machine translation and summarization. The BLEU score is calculated as follows:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{2.12}$$

where BP is the brevity penalty, w_n are weights (usually uniform), and p_n is the precision for n-grams.

– **Recall-Oriented Understudy for Gisting Evaluation (ROUGE)** [16]: Measures overlap of n-grams, word sequences, and word pairs, especially for summarization tasks. The following is the formula to compute this metric:

$$\text{ROUGE-N} = \frac{\sum_{S \in \{\text{Reference}\}} \sum_{\text{gram}_n \in S} \text{Count}_{\text{match}}(\text{gram}_n)}{\sum_{S \in \{\text{Reference}\}} \sum_{\text{gram}_n \in S} \text{Count}(\text{gram}_n)} \tag{2.13}$$

where $\text{Count}_{\text{match}}(\text{gram}_n)$ counts the matching n-grams in the generated text, and $\text{Count}(\text{gram}_n)$ counts the total n-grams in the reference.

– **BERTScore**: Uses contextual embeddings from BERT to compute semantic similarity between generated and reference texts, providing a more nuanced evaluation than surface-level overlap. It is defined as:

$$\text{BERTScore} = \frac{1}{|X|} \sum_{x \in X} \max_{y \in Y} \text{sim}(x, y) \tag{2.14}$$

where X and Y are token embeddings from candidate and reference, and sim is typically cosine similarity.

– **Metric for Evaluation of Translation with Explicit ORdering (METEOR)**: Considers synonymy and stemming, aligning generated text with reference answers based on semantic similarity. The METEOR score is given by:

$$\text{METEOR} = F_{\text{mean}} \cdot (1 - \text{Penalty}) \tag{2.15}$$

where F_{mean} is a harmonic mean of unigram precision and recall, and Penalty is based on alignment fragmentation.

Evaluation Tools and Frameworks

A variety of tools and frameworks are available to facilitate the evaluation of retrieval and generation systems, we here cite the ones that used in the work done in this thesis:

- **trec_eval**: A tool for evaluating retrieval results using QRELS as comparison since they are ground-truth relevance judgements created by human assessors, supporting metrics such as NDCG, MAP, and MRR. The following is an example of output of the command:

```
$ ./trec_eval qrels.json results.json
```

Code 2.1: *trec_eval* sample command for evaluating a sample result file using QRELS

being run on a retrieval result file:

Metric	Query Set	Value
runid	all	r_comp24_lucene_segmented_monot5
num_q	all	301
num_ret	all	30100
num_rel	all	68780
num_rel_ret	all	9093
map	all	0.1120
gm_map	all	0.0572
Rprec	all	0.1393
bpref	all	0.1333
recip_rank	all	0.8723
iprec_at_recall_0.00	all	0.8983
iprec_at_recall_0.10	all	0.4829
iprec_at_recall_0.20	all	0.1597
iprec_at_recall_0.30	all	0.0473
iprec_at_recall_0.40	all	0.0161
iprec_at_recall_0.50	all	0.0083
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	all	0.7847
P_10	all	0.7492
P_15	all	0.7116
P_20	all	0.6865
P_30	all	0.6060
P_100	all	0.3021
P_200	all	0.1510
P_500	all	0.0604
P_1000	all	0.0302

Table 2.1: Evaluation metrics computed with *trec_eval* tool for a sample retrieval result file using QRELS

- **Nugget Assignment**: It can be done using the *AutoNuggetizer* framework [17], which assesses whether a system-generated answer includes key factual elements (”nuggets”) relevant to a query. Nuggets are concise information units, categorized as ”vital” (essential) or ”okay” (beneficial but not critical) and the evaluation involves checking if these nuggets are fully, partially, or not at all present in the answer. Metrics used include:

– **All Score**: Overall coverage of all nuggets.

$$s_i = \begin{cases} 1 & \text{if assignment = support} \\ 0.5 & \text{if assignment = partial\_support} \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

$$A = \frac{\sum_i s_i}{N_{\text{nuggets}}} \tag{2.17}$$

– **Strict Score**: Coverage focusing solely on vital nuggets.

$$ss_i = \begin{cases} 1 & \text{if assignment = support} \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

$$A_{\text{strict}} = \frac{\sum_i ss_i}{N_{\text{nuggets}}} \tag{2.19}$$

– **Vital Score**: Same as All Score, but computed only over vital nuggets.

$$s_i^v = \begin{cases} 1 & \text{if assignment = support} \\ 0.5 & \text{if assignment = partial\_support} \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

$$V = \frac{\sum_i s_i^v}{|n_v|} \tag{2.21}$$

– **Vital Strict Score**: Strict version of Vital Score, only full support of vital nuggets counts.

$$ss_i^v = \begin{cases} 1 & \text{if assignment = support} \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

$$V_{\text{strict}} = \frac{\sum_i ss_i^v}{|n_v|} \tag{2.23}$$

The framework provides three main scripts: one for nugget creation, one for nugget assignment, and one for computing the evaluation metrics.

- **Support Evaluation**: It's a generation evaluation that works by assign-

ing values "Full Support", "Partial Support", and "No Support" (using a SOTA LLM, for example GPT-4o) to assess how an answer to a question is backed by citations. Metrics include:

  - **Weighted Precision**: Measures the proportion of answer sentences supported by cited sources, assigning more weight to fully supported sentences.

$$\text{Weighted Precision} = \frac{\sum_{i,j} s(a_i, d_j)}{\text{count}(\{a_i, d_j\})} \tag{2.24}$$

    where $s(a_i, d_j) \in \{0, 0.5, 1\}$ depending on support level, a_i is an answer sentence, d_j is a cited document.

  - **Weighted Recall**: Assesses how comprehensively the cited sources are utilized to support the answer, with greater weight for fully supported content.

$$\text{Weighted Recall} = \frac{\sum_{i,j} s(a_i, d_j)}{\text{count}(\{a_i\})} \tag{2.25}$$

- **Fluency Assessment**: Another generation evaluation metric is fluency, which evaluates the coherence and grammatical correctness of generated text and can be performed in the same way as Support Assessment, but instead assigns a score based on the fluency of the text.

2.1.4 Retrieval-Augmented Generation

RAG [18] is an approach that combines IR techniques with generative LLMs to enhance the quality and factual accuracy of generated text. Given that traditional language models are limited by the information encoded in their parameters at training time, which can quickly become outdated or insufficient for specialized queries, RAG addresses this issue by retrieving relevant documents or passages from an external knowledge source (such as a large text corpus or database) and conditioning the generation process on this retrieved context. The typical RAG workflow involves two main stages: first, a retriever component selects a set of relevant documents based on the input query; then, a generator (usually a LLM) produces an answer or output conditioned on both the query and the retrieved documents (by providing it a correct prompt), enabling the system to ground its responses in up-to-date or domain-specific information,

improving both relevance and factual correctness. It has become increasingly popular in recent years, especially for tasks like Open-Domain QA, summarization, and knowledge-intensive generation, as it allows models to dynamically access external knowledge without the need for costly retraining, with ongoing research continuing to develop new techniques and approaches for more efficient retrieval, better integration between retrieval and generation, and improved evaluation of these systems.

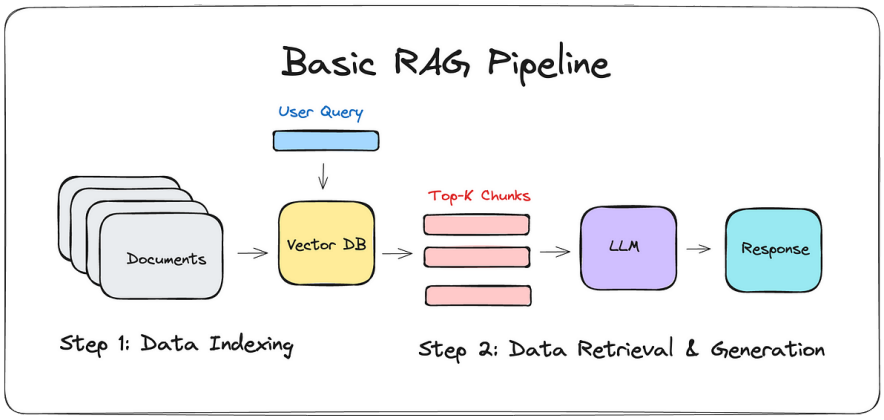

Figure 2.4: Overview of the RAG approach.

Parametric Retrieval-Augmented Generation

PRAG is a novel paradigm (January 2025, [19]) introduced to overcome the limitations of traditional in-context RAG methods: while the conventional RAG approach appends retrieved documents to the model's input context (the prompt), with PRAG there is a direct integration of external knowledge into the parameters of a LLM, specifically by injecting document representations into the Feed-Forward Neural Network (FFN) layers of the model, making the LLM able to process retrieved information as if it were part of its internal knowledge, reducing dependency on long context windows and improving both computational efficiency and reasoning quality.

In particular, PRAG consists of two primary stages:

- **Offline document parameterization phase**: Each document is transformed into a compact parametric representation (that is then stored and associated with its corresponding document) using data augmentation (e.g., document rewriting and QA pair generation) followed by

fine-tuning lightweight low-rank adapters (such as LoRA) that encode the knowledge into the model's FFN weights.

- **Online inference phase**: The system retrieves the most relevant documents based on the input query, loads and merges the corresponding parametric representations into the LLM's FFN layers, and uses the updated model—now temporarily enriched with the relevant knowledge—to generate the output without requiring the documents to be appended to the input prompt.

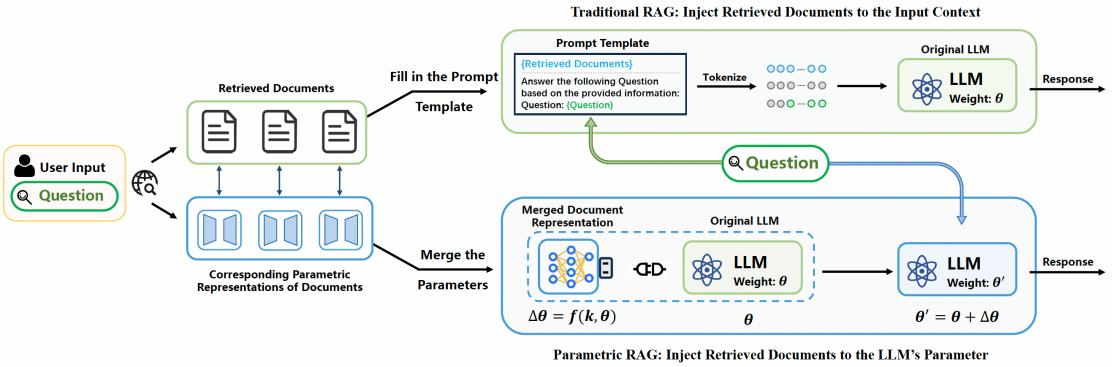

Figure 2.5: Overview of the PRAG approach compared to traditional RAG pipeline.

Experimental results demonstrate that this approach outperforms standard and advanced RAG baselines on several benchmarks, particularly in tasks requiring multi-hop reasoning or complex fact integration and, since it is compatible with traditional in-context RAG, combining both methods can yield additional performance gains. Despite its need for an offline parameterization phase, it offers long-term efficiency benefits in large-scale deployments, making it a promising direction for future systems.

2.2 TREC RAG 2024 Track

The TREC RAG 2024 Track was established to advance research on systems that combine classical IR with LLMs: in this framework, systems follow the typical RAG pipeline by first retrieving relevant text segments from a large corpus and then by using those segments to ground or augment an LLM's generated answers. The track is structured into three related tasks using the new MS MARCO

v2.1 dataset, providing a unified benchmark to analyze retrieval and generation components both separately and together in end-to-end RAG systems. The creators of the track provided a set of prebuilt indexes (both sparse and dense) of the MS MARCO v2.1 collection (both segmented and complete) and methods/baselines accessible via the *Pyserini* framework to facilitate participation and experimentation.

2.2.1 Retrieval Task

In the R task, participants perform ad-hoc ranking of text segments: each system receives a list of non-factoid query topics and must return the top 100 most relevant segment IDs for each query in a standard TREC run file. The follwing is the structure of a TREC run file with a single sample entry:

Topic ID	Fixed	Doc ID	Rank	Score	Run ID
2024-44544	Q0	msmarco_v2.1_doc_42_581410495#3_1140003275	1	0.9949001669883728	r_comp24_lucene_segmented_monot5

Table 2.2: Sample entry in a TREC run file for the R Task

2.2.2 Augmented Generation Task

The Augmented Generation (AG) task focuses on answer generation given retrieved context. Here systems receive the same topics and segment collection as in the R task, plus a fixed list of top-k relevant segments per topic (provided by the organizers) and, using this context, they must generate a grounded, long-form answer (up to 400 words), broken into sentences with citations to supporting segments. Since retrieval is fixed, this task emphasizes the quality, informativeness, and faithfulness of the generated answer. This is an example of a generated answer (a JSON file) in the AG task:

```
{
    "run_id": "ag_zephyr_load_prag_contexts_G_comp24
                    _lucene_segmented_monot5",
    "topic_id": "2024-134964",
    "topic": "what do democrats think about abortion",
    "references": [
        "msmarco_v2.1_doc_51_296614086#0_670238825",
        "msmarco_v2.1_doc_51_296614086#2_670243117",
        "msmarco_v2.1_doc_08_1572680298#10_2839677033",
```

```
        "msmarco_v2.1_doc_10_1185732870#9_1889669859",
        "msmarco_v2.1_doc_22_400876093#16_975171449",
        "msmarco_v2.1_doc_05_1248415924#5_2379624439",
        "msmarco_v2.1_doc_25_133843467#4_271952790",
        "msmarco_v2.1_doc_22_400876093#15_975169956",
        "msmarco_v2.1_doc_57_164119664#11_359110893",
        "msmarco_v2.1_doc_30_520836676#0_1179098859"
    ],
    "response_length": 111,
    "answer": [
        {
            "text": "The Democratic Party believes firmly in a woman's
            right to decide whether or not to abort a pregnancy.",
            "citations": [2]
        },
        {
            "text": "Democratic views on abortion strongly support Roe
            vs. Wade, and include a woman being given the right to
            abort a pregnancy regardless of whether or not she is able
            to pay for it.",
            "citations": [2]
        },
        {
            "text": "They aim to make abortion more rare, not more
            dangerous, supporting contraceptive research, family
            planning, comprehensive family life education, and
            policies that support healthy childbearing.",
            "citations": [0]
        },
        {
            "text": "While most Americans support keeping Roe vs.
            Wade as the ruling, opinions on abortion have grown
            increasingly divided.",
            "citations": [1]
        },
        {
```

```
            "text": "Republican voters usually believe that abortion
            should be heavily regulated and opposed.",
            "citations": [3]
        }
    ]
}
```

2.2.3 Retrieval-Augmented Generation Task

The RAG task requires a combiation of the previous task, thus to provide a full end-to-end retrieval and generation pipeline. Participants are given the same data as in the R and AG tasks, but they must first retrieve relevant segments for each topic and then generate a grounded answer using those segments. The system must retrieve relevant content, generate a coherent answer with citations, and map any retrieved chunks to the official segment IDs for evaluation: this setup mimics very well industrial RAG pipelines. The output format is the same as in the AG task.

2.2.4 Evaluation Metrics and Tools

Evaluation is done with the tools and metrics described in the previous section, with the addition of manual evaluation of the generated answers by human assessors for all three evaluation metrics (Nugget Assignment, Support Evaluation, and Fluency Assessment).

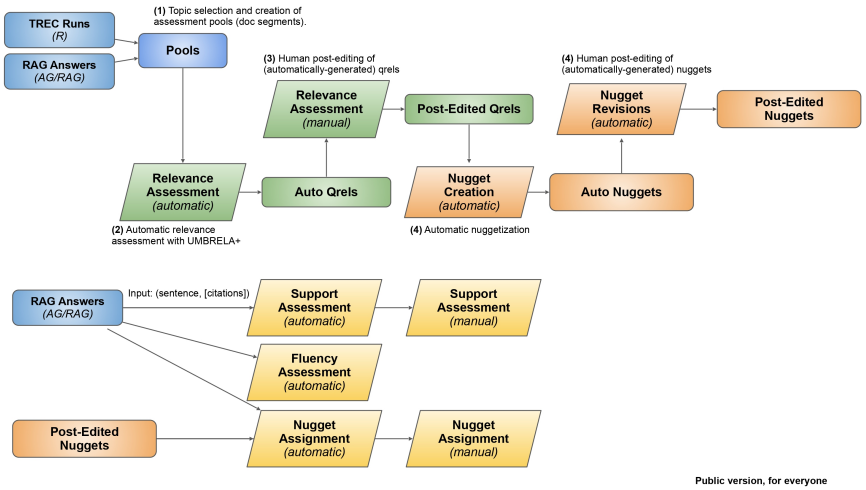

Figure 2.6: TREC RAG 2024 Track Evaluation Process

3 Methodology

3.1 RAG System Architecture

The RAG system developed for this thesis is implemented primarily in Python, leveraging a modular architecture that integrates several SOTA libraries and frameworks. The system is designed to support flexible experimentation with retrieval and generation strategies for Open-Domain QA. At a high level, the **RAG system** consists of two main components: the first one is the **retrieval module**, responsible for efficiently searching a large corpus of documents to identify relevant passages for a given query. This is achieved using the **Pyserini** toolkit, which provides a Python interface to the Lucene search engine via `LuceneSearcher`, supporting traditional retrieval methods, or the **FAISSSearcher** and the **HybridSearcher** for more advanced and neural retrieval techniques. Both searchers are configured to accept as input a query and an index on which to operate: in this thesis case, the index is built on the MS MARCO Segment v2.1 collection and it is provided by the creators of the Track (many other prebuilt indexes are available, also in various formats). For re-ranking retrieved passages, transformer-based models such as **MonoT5** and **MonoBERT** are employed, utilizing the HuggingFace Transformers library. The second module is the **generation module** and is built on top of transformer architectures, including encoder-decoder models (**T5 Large** and **T5 XL**) and decoder-only models (**LLAMA2**, **LLAMA3** [20] and **Zephyr** [21]). These models are accessed and fine-tuned using the HuggingFace Transformers library, with

support for efficient training and inference provided by PyTorch, bitsandbytes (for quantization), and Accelerate (for distributed and mixed-precision training). Tokenization and text preprocessing are handled using Natural Language Toolkit (NLTK) [22] and HuggingFace tokenizers, while data manipulation and experiment tracking utilize pandas [23] and tqdm. The system is highly configurable, allowing the user to select datasets, retrieval indices, rerankers, generation models, architectures, prompts and many other options at runtime. This flexibility is achieved through a configuration module and utility functions that manage experiment setup, data loading, and evaluation. The overall workflow involves retrieving relevant passages for a query, optionally re-ranking them, and then generating an answer conditioned on the retrieved context using the selected language model. The architecture supports both local and API-based (OpenRouter, Google AI Studio) model inference, enabling experimentation with a wide range of models and resources.

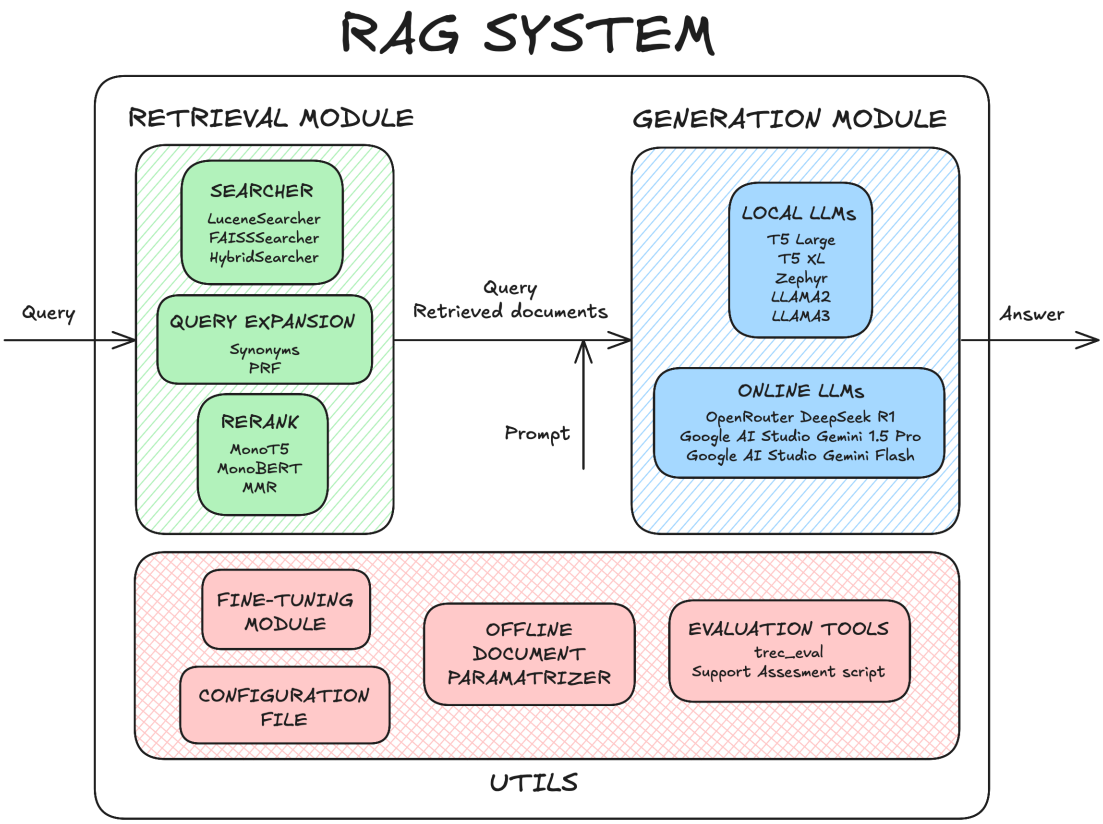

Figure 3.1: Overview of the proposed RAG System Architecture

3.2 Retrieval Module

The **retrieval module** is a core component of the RAG system, responsible for efficiently identifying and ranking relevant passages from a large document corpus in response to a user query. This part is designed to be modular and extensible, supporting multiple retrieval backends and advanced re-ranking and query expansion strategies.

3.2.1 Searching

The system supports multiple types of indices to enable flexible retrieval strategies using the three different provided searchers from the **Pyserini** library:

- **LuceneSearcher**: This searcher is used for traditional retrieval methods, specifically the Lucene BM25 algorithm and operates on a prebuilt index (that must be an inverted index) that contains documents or passages from a corpus. It has two hyperparameters that can be tuned: *k1* and *b*, which control the term frequency saturation and document length normalization, respectively.
- **FAISSSearcher**: This searcher is used for dense vector similarity search, leveraging the FAISS library to perform efficient nearest neighbor search on embeddings.
- **HybridSearcher**: This searcher combines both Lucene and FAISS indices, allowing for hybrid retrieval strategies that can leverage both traditional keyword-based and dense vector search.

Note that indexes can be prebuilt ones (provided by **Pyserini**) or custom ones built automatically using the same library or manually: in all cases they must be stored locally and can require significant disk space depending on the corpus size and index type. Queries data is loaded into a custom **ExtendedDataset** class that handles the splitting into training and test sets with a percentage parameter (used for testing purposes) and fed into a **DataLoader** instance (from the **torch** module) to allow for efficient batching and parallel processing. Given a user query, the retrieval module can search top k relevant passages from the selected index type, with searchers created to accept both the query and the index/corpus to operate on, with support for either full documents or segmented

passages, depending on the corpus granularity. In the end, the output format is a Tab-Separated Values (TSV) file containing, for each query/topic, the retrieved passages IDs with the corresponding rank and score (and the run ID for tracking purposes) as for TREC standard requirements.

3.2.2 Query Expansion

The retrieval module also can use advanced query expansion techniques to enhance recall:

- **Synonym Expansion**: The query can be expanded with contextually relevant synonyms using the WordNet library [24] and Part of Speech (POS) tagging by identifying key terms in the query and adding synonyms that are semantically related after the original term, increasing the likelihood of matching relevant passages. This can be controlled with a parameter *max_synonyms* that specifies the maximum number of synonyms to add per identified keyword. For example, the query:

  > *how does religion show in public school*

  will be expanded to something like:

  > *how does religion faith belief show in public school educational institution learning environment*

  The implementation of the synonym expansion function is shown below:

```
    def get_wordnet_pos(treebank_tag):
        mapping = {'J': wn.ADJ, 'V': wn.VERB, 'N': wn.NOUN, 'R':
    wn.ADV}
        return mapping.get(treebank_tag[0], None)

    def expand_query(query, max_synonyms=2):
        lemmatizer = WordNetLemmatizer()
        tokens = word_tokenize(query.lower())
        pos_tags = pos_tag(tokens)
        stop_words = set(stopwords.words('english')) | set(
    string.punctuation)
        expanded = []
        for token, tag in pos_tags:
            if len(token) <= 2 or not token.isalpha() or token
    in stop_words:
```

```
                expanded.append(token)
                continue
            # Get WordNet POS tag
            wn_pos = get_wordnet_pos(tag)
            if not wn_pos:
                expanded.append(token)
                continue
            # Lemmatize the token
            lemma = lemmatizer.lemmatize(token, pos=wn_pos)
            # Get synonyms from the most common synset
            synsets = wn.synsets(lemma, pos=wn_pos)[:1]  # Use
only the most common synset
            synonyms = []
            for syn in synsets:
                for lemma in syn.lemmas():
                    synonym = lemma.name().lower().replace('_',
' ')
                    if synonym == token or ' ' in synonym:
                        continue  # Skip multi-word synonyms and
 duplicates
                    if syn.pos() == 'n' and token.endswith('s')
and not synonym.endswith('s'):
                        synonym += 's'  # Pluralize if original
was plural
                    synonyms.append(synonym)
            # Add original token and its synonyms
            expanded.append(token)
            expanded.extend(synonyms[:max_synonyms])
        return ' '.join(expanded)
```

Code 3.1: Synonym Expansion function

- **PRF Expansion**: this expansion is implemented by analyzing the retrieved documents for the initial query and extracting the most informative terms. Specifically, after the first retrieval step, the system computes the TF-IDF scores for terms in the top passages, selects the top n terms (controlled by the *n_terms* parameter), and appends them to the original query to form an expanded query. This expanded query is then used to perform a second retrieval to allow the retrieval module to dynamically adapt the query based on the actual content of the most relevant documents, effectively simulating user feedback in an automated way. The following code snippet shows the implementation of the PRF expansion function:

```
    def expand_query_with_feedback(query, top_docs, n_terms=3):
        # Extract text from top documents
        texts = [doc['segment'] for doc in top_docs]

        # Compute TF-IDF to find important terms
        vectorizer = TfidfVectorizer(stop_words='english')
        tfidf = vectorizer.fit_transform(texts)

        # Get top terms based on TF-IDF scores
        feature_names = vectorizer.get_feature_names_out()
        sorted_tfidf = np.argsort(tfidf.sum(axis=0).A1)[::-1]  # Sort by TF-IDF scores
        expansion_terms = [feature_names[i] for i in sorted_tfidf[:n_terms]]

        # Combine original query with expansion terms
        expanded_query = query + " " + " ".join(expansion_terms)
        return expanded_query
```

Code 3.2: PRF Expansion function

3.2.3 Re-Ranking

To improve retrieval quality, the top retrieved passages can be re-ranked using transformer-based models:

- **MonoT5**
- **MonoBERT**

Re-ranking is performed using the HuggingFace Transformers library, and the models can be run on Central Processing Unit (CPU) or GPU (with acceleration and quantization support with respectively *Accelerate* and *bitsandbytes* libraries). as available. The re-ranking process can be applied after initial retrieval to reorder the top-k candidates based on neural relevance scores.

An additional re-ranking technique implemented in the system is **MMR**, used to balance the relevance and diversity of the retrieved results and controlled by a parameter λ whose values ranges from 0 (more diversity) to 1 (more relevance). After the initial retrieval, the system computes the similarity between the query and each candidate document (the number of candidates is controlled by the *top_k* parameter), as well as the similarity among the documents

themselves and the MMR algorithm iteratively selects documents that are both highly relevant to the query and as dissimilar as possible from those already selected, thus reducing redundancy and increasing the diversity of the top results. Below, the implementation of the MMR re-ranking function is shown:

```
def mmr_rerank(query, docs, lambda_param=0.7, top_k=20):
    # Vectorize query and documents
    vectorizer = TfidfVectorizer()
    doc_vectors = vectorizer.fit_transform(docs)
    query_vector = vectorizer.transform([query])

    # Compute relevance scores
    relevance_scores = cosine_similarity(query_vector, doc_vectors).
    flatten()

    selected = []
    remaining = set(range(len(docs)))

    while len(selected) < top_k:
        mmr_scores = []
        for idx in remaining:
            # Compute max similarity to already selected docs
            if selected:
                max_sim = max([cosine_similarity(
                    doc_vectors[idx],
                    doc_vectors[s]
                ) for s in selected])
            else:
                max_sim = 0

            # Compute MMR score
            mmr = lambda_param * relevance_scores[idx] - (1 -
    lambda_param) * max_sim
            mmr_scores.append((mmr, idx))

        # Select document with highest MMR score
        best = max(mmr_scores)
        selected.append(best[1])
        remaining.remove(best[1])

    return selected
```

Code 3.3: MMR re-ranking function

3.3 Generation Module

The **generation module** is responsible for producing natural language answers to user queries, conditioned on the context retrieved by the retrieval module. This component is implemented using SOTA transformer-based LLMs, supporting both encoder-decoder and decoder-only architectures. The system is designed to be highly modular and configurable, allowing for experimentation with different models, prompts, and generation strategies.

3.3.1 Supported Models and Frameworks

The module supports a variety of LLMs, accessed and managed using the HuggingFace Transformers library. Efficient training and inference are further supported by PyTorch, with optional use of *bitsandbytes* for quantization and *Accelerate* for distributed and mixed-precision training. All models are locally hosted and can be loaded from the HuggingFace Hub as base models and fine-tuning checkpoints can also be used if available in the machine: this is particularly useful since it is possible to fine-tune the models (which we will discuss about in this section) and then use the saved configurations. A possibility to use external APIs is also available, such as OpenRouter and Google AI Studio, which allows for the use of models that are not hosted locally, but require API keys for access and only have a limited number of free requests per day or other restrictions. The available models include:

- **Local models**:
  - **Zephyr** (stablelm-zephyr-3b)
  - **LLAMA2** (Llama-2-7b-chat-hf)
  - **LLAMA3** (Meta-Llama-3-8B)
  - **T5 Large** (flan-t5-large)
  - **T5 XL** (flan-t5-xl)
- **Hosted models**:
  - OpenRouter **DeepSeek R1** (deepseek-r1:free)
  - Google AI Studio **Gemini 1.5 Pro** (gemini-1.5-pro)
  - Google AI Studio **Gemini Flash** (gemini-1.5-flash)

3.3.2 Fine-Tuning

The fine-tuning process for the language models is managed by a dedicated module, that allows for both fine-tuning a model (from the local supported models pool) from scratch or loading a previously saved checkpoint for further training. The process begins by preparing the training dataset using a custom **WebGPTDataset** class (loaded into a PyTorch **DataLoader** for efficient batching) that loads the *WebGPT* dataset containing QA pairs, that gets split into training and validation sets based on a configurable test size parameter. For parameter-efficient fine-tuning, the script applies **LoRA** via the **peft** library, freezing the main model weights and enabling gradients only for the LoRA parameters and uses AdamW as the optimizer with a fully configurable input parameter *training_arguments* that allows the user to set various training parameters such as learning rate, batch size, number of epochs, and evaluation strategy. Fine-tuning is performed following the configured training arguments, with the model being trained on the training dataset and evaluated on the validation set using a custom evaluation function that computes the chosen metric between the following options:

- **EM**
- **ROUGE**
- **BLEU**
- **BERT**
- **METEOR**

Loss is computed during training, and the model configuration is saved at the end of an epoch if the evaluation score improves compared to the previous best score: this is not always the best approach, since the model can overfit on the training data, but it is a good starting point for further fine-tuning and experimentation.

3.3.3 Prompting and Context Integration

The generation process is prompt-driven: a system prompt and a user prompt are constructed for each query, with the user prompt optionally

including the retrieved context passages as the system supports both citation-aware and citation-free prompting, enabling the generation of answers that can reference specific passages from the retrieved context when required. If present, the context is formatted and inserted into the prompt according to the selected architecture and experiment configuration. Many prompt templates are available, for example:

```
You are an assistant that provides answers and the source of the
answer. I will give a question and several context texts about
the question. Choose the context texts that are most relevant to
the questions and based on them, give a short answer to the
question. You must provide in-line citations to each statement
in the answer from the context. The citations should appear as
numbers within brackets [] such as [1], [2] based on the given
contexts. A statement may need to be supported by multiple
contexts and should then be cited as [1] [2].
```

3.3.4 Batch Processing and Experiment Tracking

To facilitate large-scale experiments, the generation module processes queries in batches, with configurable batch sizes and limits for testing purposes. Each experiment run is tracked using a unique run identifier that encodes the selected configuration (model, dataset, architecture, etc.), ensuring reproducibility and organized result storage.

3.3.5 Output Formatting and Post-processing

The generated answers, along with relevant metadata (such as references and response length), are post-processed and formatted and finally saved in structured JSON output files in TREC required format for subsequent evaluation.

3.4 PRAG Approach

In this project we also tested the **PRAG** approach and created a separate module for its offline document parametrization phase, while also adapting the generation module to utilize the generated document-specific adapters. In the standard workflow, the entire corpus is parametrized but, due to the large size

of the dataset in this thesis (MS MARCO v2.1), a practical modification was introduced: instead of parametrizing the entire corpus, for each query/topic in the TREC RAG 2024 track, the script retrieves the top-k relevant documents using the retrieval module and, for each of these documents, the following steps are performed (after loading the chosen LLM from the local models pool):

- Extract the document content (passage).
- Generate n (controlled by the *n_rewrites* parameter) rewrites of the passage using the language model, each in a different style but preserving the original information.
- Generate m (controlled by the *n_qa* parameter) synthetic QA pairs based on the passage content.
- Use the rewrites and QA pairs to fine-tune a document-specific LoRA adapter, following the PRAG methodology.

The following is the code utilized for the generation of rewrites and QA pairs (it also uses several fallback strategies in case the generation fails or produces no content):

```
def generate_qa_and_rewrites(passage, tokenizer, model, n_rewrites=2,
    n_qas=3):
    # Move input tensors to model device
    device = model.device
    # Define prompt templates similar to PRAG
    rewrite_prompt = f"""Rewrite the following passage in a different
    style while preserving all the information:
    Passage:
    {passage}
    Rewritten passage:"""
    qa_prompt = f"""Generate a question and answer pair based on the
   following passage. The question should be prefixed with 'Q:' and
   the answer with 'A:'.
    Passage:
    {passage}
    Q:"""
    # Generate rewrites
    rewrites = []
    for i in range(n_rewrites):
        inputs = tokenizer(rewrite_prompt, return_tensors="pt",
   padding=True).to(device)
```

```
        with torch.no_grad():
            outputs = model.generate(
                **inputs,
                max_new_tokens=256,
                temperature=0.7,
                do_sample=True,
                top_p=0.9
            )
        decoded = tokenizer.decode(outputs[0], skip_special_tokens=True)
        # Extract only the generated portion
        if "Rewritten passage:" in decoded:
            rewritten_text = decoded.split("Rewritten passage:")[1].strip()
        else:
            rewritten_text = decoded.replace(rewrite_prompt, "").strip()
        if rewritten_text:
            rewrites.append(rewritten_text)
    # Always include the original passage as one of the rewrites
    if passage not in rewrites:
        rewrites.append(passage)
    # Generate QA pairs
    qa_pairs = []
    for i in range(n_qas):
        inputs = tokenizer(qa_prompt, return_tensors="pt", padding=True).to(device)
        with torch.no_grad():
            outputs = model.generate(
                **inputs,
                max_new_tokens=128,
                temperature=0.7,
                do_sample=True,
                top_p=0.9
            )
        decoded = tokenizer.decode(outputs[0], skip_special_tokens=True)
        # Process the generated text to extract Q&A
        if "Q:" in decoded:
            try:
                # Extract just the generated portion
                generated_text = decoded.replace(qa_prompt, "").strip()
```

```
                # Add Q: prefix if it was removed in the extraction
                if not generated_text.startswith("Q:"):
                    generated_text = "Q:" + generated_text
                # Split into question and answer
                question_part = generated_text.split("A:")[0].replace("Q:", "").strip()
                # Check if we have an answer part
                if "A:" in generated_text:
                    answer_part = generated_text.split("A:")[1].strip()
                    if question_part and answer_part:
                        qa_pairs.append((question_part, answer_part))
                else:
                    # Generate an answer for this question
                    answer_prompt = f"""Passage:
                    {passage}
                    Question: {question_part}
                    Answer:"""
                    answer_inputs = tokenizer(answer_prompt, return_tensors="pt", padding=True).to(device)
                    with torch.no_grad():
                        answer_outputs = model.generate(
                            **answer_inputs,
                            max_new_tokens=128,
                            temperature=0.3,
                            do_sample=True,
                            top_p=0.9
                        )
                    answer_decoded = tokenizer.decode(answer_outputs[0], skip_special_tokens=True)
                    # Extract the answer
                    answer_text = answer_decoded.split("Answer:")[1].strip() if "Answer:" in answer_decoded else answer_decoded.replace(answer_prompt, "").strip()
                    if question_part and answer_text:
                        qa_pairs.append((question_part, answer_text))
            except Exception as e:
                print(f"Error extracting QA pair: {e}")
                continue
    # Ensure we have at least some content
    if not rewrites:
        rewrites = [passage]  # Use original if no rewrites generated
    if not qa_pairs:
```

```
91          # Create a simple fallback QA pair
92          first_sentence = passage.split('.')[0].strip() + '.'
93          qa_pairs = [(f"What does the passage say about {
    first_sentence.split()[1] if len(first_sentence.split()) > 1 else
    'this topic'}?", first_sentence)]
94      print(f"Generated {len(rewrites)} rewrites and {len(qa_pairs)} QA
         pairs")
95      return rewrites, qa_pairs
```

Code 3.4: Code for generating rewrites and QA pairs in the PRAG approach

Saved configurations are mapped with the correct IDs of queries/topics and document IDs. This selective parametrization strategy enables the application of PRAG to large-scale datasets by restricting computational resources to the most relevant documents per query, while preserving the core idea of document-specific adaptation. The chosen model is then fine-tuned with the generated rewrites and QA pairs, creating a document-specific adapter that captures the nuances of the passage, with loss being tracked and saved for analysis for each step and epoch during the training phase. Finally, in the generation module, two extra configuration parameters are added: one is to enable the possibility of loading the document-specific adapters (if they exist, else the answer is processed normally) during the generation phase and the other is to specify if the contexts must be also integrated into the LLM prompt or not.

3.5 Evaluation

The evaluation of the system retrieval and generation runs can be performed almost exhaustively using tools and scripts from the Track creators and from the system's repository.

3.5.1 Retrieval Evaluation

For retrieval runs, the official `trec_rag` evaluation tool provided by TREC is used: the tool is available inside the system's repository and can be run with the command 2.1 shown in the previous chapter.

3.5.2 Generation Evaluation

For generation runs, several evaluation strategies are available to reproduce or emulate the official TREC RAG 2024 Track evaluation process, which includes both manual and automatic validation methods. Two important notes are to be made before discussing about the evaluation methods:

- The manual evaluation of answers can be performed, but it is not recommended as it is not reproducible or directly comparable to official track results.
- One can use the validator script from the Ragnarok repository (a tool created by the track organizers to make publicly available a baseline RAG system with a set of prebuilt indexes and models, all fully configurable and compatible with the track requirements [25]) to check if generated answers are structured correctly as for the track requirements (e.g. answers must be of length less than 400 tokens, file must have a precise structure, etc.).
- **Nugget Assignment**: This evaluation can be performed using the **AutoNuggetizer** tool, which contains all necessary scripts for mapping answer content to reference nuggets and compute the needed metrics.
- **Support Assessment**: It can be reproduced (though not officially the same as the track, since the official code was not released) with a script inside the project's repository that emulates the official methodology by calling OpenAI's API completion endpoint with the GPT-4o model and the same prompt used in the track to assign the correct values to answers and finally compute the needed metrics. The following is the prompt used:

```
In this task, you will evaluate whether each statement
is supported by its corresponding citations. Note
that the system responses may appear very fluent and
well-formed, but contain slight inaccuracies that
are not easy to discern at first glance. Pay close
attention to the text. You will be provided with a
statement and its corresponding passage which the
statement cites. It may be helpful to ask yourself
whether it is accurate to say "according to the
citation ..." with the statement following this
```

> *phrase. Be sure to check all of the information in the statement. You will be given three options: • Full Support: All of the information in the statement is supported in the citation. • Partial Support: Some parts of the information are supported in the citation, but other parts are missing. • No Support: The citation does not support any part of the statement. Please provide your response based on the information in the citation. If you are unsure, use your best judgment. Respond as either "Full Support", "Partial Support", or "No Support" with no additional information. Statement: ---statement--- Citation: ---citation_text---*

where *statement* is the generated answer and *citation_text* is the retrieved context used to generate the answer. Again, like for manual evaluation for generation runs, this evaluation is not truly comparable to the official track results, thus the comparisons with other track participants that will be made in the next chapter are not based on the official support results but instead on the results computed with the script on both the proposed system's and the other participants' runs.

- **Fluency Assessment**: Fluency assessment could not be reproduced, as the TREC organizers did not release any code or paper to explain how they performed it.

This evaluation pipeline ensures that both retrieval and generation components are assessed using standardized and reproducible methods wherever possible, facilitating fair comparison with other systems and track participants.

4

Experiments

4.1 Retrieval Experiments

In this section, we describe the retrieval experiments conducted as part of this work: they are all based on the MS MARCO v2.1 segmented collection and use the official TREC topic queries. Tests were done by retrieving 5, 10, 20 and 100 top documents per query but we report, as per TREC requirements, only experiments with 100 retrieved documents. Dense and hybrid retrieval approaches were not included due to computational constraints (time and physical memory) and unreliable results, thus the utilized retrieval methods in these experiments are all based on the **BM25 LuceneSearcher** with the provided full sparse inverse index of the segmented dataset.

4.1.1 Baseline

The baseline retrieval experiment (**giaant_lucene_monot5**) used the default values of the searcher class and, after the retrieval phase, re-ranking is applied using **MonoT5**. **MonoBERT** re-ranker was also tested but its result were way worse than **MonoT5**, so the final decision was to just discard this option and just continue with the T5 based re-ranker.

Metric	Query Set	Value
runid	all	r_comp24_lucene_segmented_monot5
num_q	all	301
num_ret	all	30100
num_rel	all	68780
num_rel_ret	all	9093
map	**all**	**0.1120**
gm_map	all	0.0572
Rprec	**all**	**0.1393**
bpref	all	0.1333
recip_rank	**all**	**0.8723**
iprec_at_recall_0.00	all	0.8983
iprec_at_recall_0.10	all	0.4829
iprec_at_recall_0.20	all	0.1597
iprec_at_recall_0.30	all	0.0473
iprec_at_recall_0.40	all	0.0161
iprec_at_recall_0.50	all	0.0083
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.7847**
P_10	**all**	**0.7492**
P_15	**all**	**0.7116**
P_20	**all**	**0.6865**
P_30	**all**	**0.6060**
P_100	**all**	**0.3021**
P_200	all	0.1510
P_500	all	0.0604
P_1000	all	0.0302

Table 4.1: Retrieval Experiment *giaant_lucene_monot5*

The results show that retrieval is very effective for the top-ranked documents (e.g., P@5 and P@10 are high), but precision decreases as more documents are retrieved while MAP and Rprec are moderate, indicating reasonable overall effectiveness across all queries, not just the top results.

4.1.2 Fine-Tuned BM25

To improve retrieval effectiveness, fine-tuning of the BM25 parameters was tested: the parameters adjusted were k_1 and b, with the default values being $k_1 = 0.9$ and $b = 0.4$ that, after tuning, were set to $k_1 = 1.2$ and $b = 0.75$. We refer to this run/experiment as **giaant_lucene_monot5_f**.

Metric	Query Set	Value
runid	all	r_comp24_lucene_segmented_monot5
num_q	all	301
num_ret	all	30100
num_rel	all	68780
num_rel_ret	all	9249
map	**all**	**0.1140**
gm_map	all	0.0586
Rprec	**all**	**0.1424**
bpref	all	0.1364
recip_rank	**all**	**0.8731**
iprec_at_recall_0.00	all	0.9002
iprec_at_recall_0.10	all	0.4841
iprec_at_recall_0.20	all	0.1777
iprec_at_recall_0.30	all	0.0550
iprec_at_recall_0.40	all	0.0168
iprec_at_recall_0.50	all	0.0109
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.7801**
P_10	**all**	**0.7399**
P_15	**all**	**0.7045**
P_20	**all**	**0.6699**
P_30	**all**	**0.6020**
P_100	**all**	**0.3073**
P_200	all	0.1536
P_500	all	0.0615
P_1000	all	0.0307

Table 4.2: Retrieval Experiment *giaant_lucene_monot5_f*

Fine-tuning the cited parameters lead to a slight but consistent improvement in all main metrics compared to the baseline: this shows that parameter optimization can help retrieval effectiveness, as seen in the higher MAP, Rprec, and recall values.

4.1.3 Query Expansion

Building on the fine-tuned BM25 setup, query expansion strategies were tested (also by combining them):

- **giaant_lucene_monot5_f_wexp**: Starting from *giaant_lucene_monot5_f* run setup, queries were expanded (before searching) with synonyms using **WordNet**.

Metric	Query Set	Value
runid	all	r_comp24_lucene_segmented_monot5
num_q	all	301
num_ret	all	30100
num_rel	all	68780
num_rel_ret	all	5296
map	**all**	**0.0642**
gm_map	all	0.0112
Rprec	**all**	**0.0851**
bpref	all	0.0828
recip_rank	all	0.7184
iprec_at_recall_0.00	all	0.7396
iprec_at_recall_0.10	all	0.2507
iprec_at_recall_0.20	all	0.0814
iprec_at_recall_0.30	all	0.0287
iprec_at_recall_0.40	all	0.0116
iprec_at_recall_0.50	all	0.0030
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.5674**
P_10	**all**	**0.5047**
P_15	**all**	**0.4534**
P_20	**all**	**0.4184**
P_30	**all**	**0.3661**
P_100	**all**	**0.1759**
P_200	all	0.0880
P_500	all	0.0352
P_1000	all	0.0176

Table 4.3: Retrieval Experiment *giaant_lucene_monot5_f_wexp*

Expanding queries with **WordNet** synonyms caused a notable drop in all metrics: this suggests that adding synonyms can introduce noise and reduce retrieval quality.

- **giaant_lucene_monot5_f_wexp_prfexp**: Building on the previous experiment, queries were further expanded using **PRF**: after the first search with queries expanded with synonyms, search was executed again by expanding queries again with relevant terms found on the top documents retrieved.

Metric	Query Set	Value
runid	all	r_comp24_lucene_segmented_monot5
num_q	all	301
num_ret	all	30100
num_rel	all	68780
num_rel_ret	all	6728
map	**all**	**0.0832**
gm_map	all	0.0092
Rprec	**all**	**0.1119**
bpref	all	0.1086
recip_rank	all	0.6235
iprec_at_recall_0.00	all	0.6611
iprec_at_recall_0.10	all	0.3240
iprec_at_recall_0.20	all	0.1445
iprec_at_recall_0.30	all	0.0616
iprec_at_recall_0.40	all	0.0277
iprec_at_recall_0.50	all	0.0148
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.5349**
P_10	**all**	**0.5066**
P_15	**all**	**0.4824**
P_20	**all**	**0.4528**
P_30	**all**	**0.4068**
P_100	**all**	**0.2235**
P_200	all	0.1118
P_500	all	0.0447
P_1000	all	0.0224

Table 4.4: Retrieval Experiment *giaant_lucene_monot5_f_wexp_prfexp*

Adding **PRF**-based expansion after WordNet expansion slightly improved results compared to using only WordNet, but performance remains much lower than the baseline and fine-tuned runs: the combination of expansions does not recover effectiveness, and MAP stays low.

- **giaant_lucene_monot5_f_prfexp**: Returning to the *giaant_lucene_monot5_f* experiment and starting from its configuration, queries were expanded only once using **PRF** (again, only after the first search).

Metric	Query Set	Value
runid	all	r_comp24_lucene_segmented_monot5
num_q	all	301
num_ret	all	30100
num_rel	all	68780
num_rel_ret	all	9554
map	**all**	**0.1190**
gm_map	all	0.0446
Rprec	**all**	**0.1526**
bpref	all	0.1480
recip_rank	all	0.8133
iprec_at_recall_0.00	all	0.8480
iprec_at_recall_0.10	all	0.4831
iprec_at_recall_0.20	all	0.2213
iprec_at_recall_0.30	all	0.0852
iprec_at_recall_0.40	all	0.0281
iprec_at_recall_0.50	all	0.0154
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.7309**
P_10	**all**	**0.6967**
P_15	**all**	**0.6611**
P_20	**all**	**0.6334**
P_30	**all**	**0.5741**
P_100	**all**	**0.3174**
P_200	all	0.1587
P_500	all	0.0635
P_1000	all	0.0317

Table 4.5: Retrieval Experiment *giaant_lucene_monot5_f_prfexp*

Using only **PRF**-based query expansion (without WordNet) resulted in the best MAP and recall so far, outperforming both the baseline and the fine-tuned BM25. This indicates that PRF is an effective query expansion strategy in this context.

- **giaant_lucene_monot5_f_prfexp_wexp**: Referring to the previous run setup, queries were further expanded with synonyms like in experiment *giaant_lucene_monot5_f_wexp*, right after PRF expansion and before searching.

Metric	Query Set	Value
runid	all	r_comp24_lucene_segmented_monot5
num_q	all	301
num_ret	all	30100
num_rel	all	68780
num_rel_ret	all	6461
map	**all**	**0.0802**
gm_map	all	0.0186
Rprec	**all**	**0.1093**
bpref	all	0.1058
recip_rank	all	0.7288
iprec_at_recall_0.00	all	0.7638
iprec_at_recall_0.10	all	0.3095
iprec_at_recall_0.20	all	0.1300
iprec_at_recall_0.30	all	0.0576
iprec_at_recall_0.40	all	0.0209
iprec_at_recall_0.50	all	0.0072
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.6020**
P_10	**all**	**0.5528**
P_15	**all**	**0.5110**
P_20	**all**	**0.4711**
P_30	**all**	**0.4188**
P_100	**all**	**0.2147**
P_200	all	0.1073
P_500	all	0.0429
P_1000	all	0.0215

Table 4.6: Retrieval Experiment *giaant_lucene_monot5_f_prfexp_wexp*

Adding WordNet expansion after **PRF** again reduced performance, similar to previous WordNet runs. The order of expansion does not in fact help as synonym expansion consistently hurts results making metrics drop significantly.

4.1.4 MMR

The following reported experiments are done using **MMR** with different values of the λ parameter to balance between diversity and relevance of the retrieved top documents.

- **giaant_lucene_monot5_f_prfexp_mmr07**: Starting from experiment *giaant_lucene_monot5_f_prfexp* (which was the most promising one in terms

of results with query expansion included), re-ranking was performed with **MMR** ($\lambda = 0.7$).

Metric	Query Set	Value
runid	all	r_comp24_lucene_segmented_monot5
num_q	all	301
num_ret	all	30100
num_rel	all	68780
num_rel_ret	all	9583
map	**all**	**0.1192**
gm_map	all	0.0443
Rprec	**all**	**0.1530**
bpref	all	0.1484
recip_rank	all	0.8100
iprec_at_recall_0.00	all	0.8441
iprec_at_recall_0.10	all	0.4794
iprec_at_recall_0.20	all	0.2299
iprec_at_recall_0.30	all	0.0853
iprec_at_recall_0.40	all	0.0283
iprec_at_recall_0.50	all	0.0154
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.7276**
P_10	**all**	**0.6910**
P_15	**all**	**0.6589**
P_20	**all**	**0.6297**
P_30	**all**	**0.5739**
P_100	**all**	**0.3184**
P_200	all	0.1592
P_500	all	0.0637
P_1000	all	0.0318

Table 4.7: Retrieval Experiment *giaant_lucene_monot5_f_prfexp_mmr07*

Applying **MMR** re-ranking ($\lambda = 0.7$) after PRF expansion maintained high MAP and recall (actually the highest between all experiments) but with a small drop on *Reciprocal Rank*: it is similar to the PRF-only run and we can notice that introducing diversity with the re-ranking algorithm does not negatively impact effectiveness.

- **giaant_lucene_monot5_f_mmr07**: At this point the decision was to drop query expansion techniques since results were not reliable enough so, building on top of, again, the second experiment (*giaant_lucene_monot5_f*), re-ranking was performed, like in the previous run, with **MMR** with the same λ value.

Metric	Query Set	Value
runid	all	r_comp24_lucene_segmented_monot5
num_q	all	301
num_ret	all	30100
num_rel	all	68780
num_rel_ret	all	9249
map	**all**	**0.1141**
gm_map	all	0.0586
Rprec	**all**	**0.1424**
bpref	all	0.1364
recip_rank	all	0.8749
iprec_at_recall_0.00	all	0.9019
iprec_at_recall_0.10	all	0.4848
iprec_at_recall_0.20	all	0.1775
iprec_at_recall_0.30	all	0.0550
iprec_at_recall_0.40	all	0.0169
iprec_at_recall_0.50	all	0.0108
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.7801**
P_10	**all**	**0.7389**
P_15	**all**	**0.7028**
P_20	**all**	**0.6698**
P_30	**all**	**0.6013**
P_100	**all**	**0.3073**
P_200	all	0.1536
P_500	all	0.0615
P_1000	all	0.0307

Table 4.8: Retrieval Experiment *giaant_lucene_monot5_f_mmr07*

MMR re-ranking ($\lambda = 0.7$) without any query expansion gave results comparable to the fine-tuned BM25: there is no significant gain, but the performance is stable in terms of MAP and other metrics and resulting documents are more diverse than other runs.

- **giaant_lucene_monot5_f_mmr05**: Since results were showing that MMR without query expansion was outperforming other runs, different values of λ were tested, here we report the run with value = 0.5.

Metric	Query Set	Value
runid	all	r_comp24_lucene_segmented_monot5
num_q	all	301
num_ret	all	30100
num_rel	all	68780
num_rel_ret	all	9249
map	**all**	**0.1141**
gm_map	all	0.0586
Rprec	**all**	**0.1424**
bpref	all	0.1364
recip_rank	all	0.8729
iprec_at_recall_0.00	all	0.9005
iprec_at_recall_0.10	all	0.4845
iprec_at_recall_0.20	all	0.1777
iprec_at_recall_0.30	all	0.0551
iprec_at_recall_0.40	all	0.0169
iprec_at_recall_0.50	all	0.0109
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.7821**
P_10	**all**	**0.7402**
P_15	**all**	**0.7050**
P_20	**all**	**0.6693**
P_30	**all**	**0.6022**
P_100	**all**	**0.3073**
P_200	all	0.1536
P_500	all	0.0615
P_1000	all	0.0307

Table 4.9: Retrieval Experiment *giaant_lucene_monot5_f_mmr05*

Results with **MMR** ($\lambda = 0.5$) showed that the retrieval achieved the best overall balance of MAP, recall, and precision among all experiments while allowing diversity in the retrieved documents, making this configuration the most effective. Given that this was the best experiment of this thesis and we will refer to it about it later in the Generation Experiments section as baseline for the generation runs.

- **giaant_lucene_monot5_f_mmr03**: We here tuned **MMR** λ to 0.3 to check if having a more aggressive diversity would improve results.

Metric	Query Set	Value
runid	all	r_comp24_lucene_segmented_monot5
num_q	all	301
num_ret	all	30100
num_rel	all	68780
num_rel_ret	all	9249
map	**all**	**0.1141**
gm_map	all	0.0586
Rprec	**all**	**0.1424**
bpref	all	0.1364
recip_rank	all	0.8745
iprec_at_recall_0.00	all	0.9017
iprec_at_recall_0.10	all	0.4846
iprec_at_recall_0.20	all	0.1777
iprec_at_recall_0.30	all	0.0551
iprec_at_recall_0.40	all	0.0169
iprec_at_recall_0.50	all	0.0109
iprec_at_recall_0.60	all	0.0000
iprec_at_recall_0.70	all	0.0000
iprec_at_recall_0.80	all	0.0000
iprec_at_recall_0.90	all	0.0000
iprec_at_recall_1.00	all	0.0000
P_5	**all**	**0.7794**
P_10	**all**	**0.7395**
P_15	**all**	**0.7043**
P_20	**all**	**0.6693**
P_30	**all**	**0.6025**
P_100	**all**	**0.3073**
P_200	all	0.1536
P_500	all	0.0615
P_1000	all	0.0307

Table 4.10: Retrieval Experiment *giaant_lucene_monot5_f_mmr03*

Lowering the **MMR** λ to 0.3 did not improve results comparing to the previous run as metrics, only Precision at various cutoffs slightly decreased compared to $\lambda = 0.5$.

4.2 RAG Experiments

In this section, we describe the experiments involving **RAG** using local LLMs. Three important notes are to be made at this point:

- To keep this thesis concise about this topic and to not have many repetitions, **AG** experiments are not cited in this document, only RAG ones (so with reference documents retrieved with our retrieval pipeline).
- All the previous cited local LLMs were used but results from models of the

LLAMA family (*LLAMA2* and *LLAMA3*) were not reliable enough since generation was very computationally expensive, thus experiments/runs using those are not reported.

- The online models described in the previous chapters were tested via API calls too and results were much better than local ones. This is to be expected since those LLMs are trained and fine-tuned massively on general tasks and can perform very well in this downstream task even if this is not their main purpose. Results from this runs are not reported too.

As already stated, the base retrieval run used to get the documents to be used as context for generation is *giaant_lucene_monot5_f_mmr05* and the optimal prompt chosen was the following:

```
You are an assistant who provides answers based on the
sources I give you and your prior knowledge, the answer must be
a single paragraph made up of statements such as Each statement
must be a complete sentence and end with the relevant context
number in square brackets [ ] or not, followed by a full stop.
If a statement is supported by more than one context, indicate
all the relevant numbers in square brackets [1][2], these
numbers must be only one of the contexts in the context section
provided only. \n Here's an example to follow : QUESTION: What
are the effects of deforestation? \n\n CONTEXTS:\n [1]
Deforestation contributes to climate change by increasing
greenhouse gas emissions.\n [2] The loss of trees leads to a
decrease in biodiversity as habitats are destroyed. \n\n ANSWER:
\n Deforestation contributes to climate change by increasing
greenhouse gas emissions [1]. The loss of trees leads to a
decrease in biodiversity as habitats are destroyed [2].
Deforestation can result in soil erosion, which negatively
impacts agriculture.
```

4.2.1 Local Base LLMs

The following are the results of runs using local LLMs in their base version, without any modification and with the suggested configurations from their providers.

- **giaant_t5l_b**: Base **T5 Large**

Nugget Results
Strict Vital Score: 0.1323
Strict All Score: 0.1319
Vital Score: 0.1801
All Score: 0.1793

Support Results
Weighted Precision: 0.8571
Weighted Recall: 0.5854

Table 4.11: Generation Experiment *giaant_t5l_b*

The base T5 Large model achieved low scores on both strict and overall nugget metrics, indicating that only a small portion of vital and all nuggets were fully or partially covered in the generated answers. However, the support metrics were relatively high, meaning that when the model did provide information, it was often well supported by the cited sources.

- **giaant_t5xl_b**: Base **T5 XL**

Nugget Results
Strict Vital Score: 0.1070
Strict All Score: 0.0725
Vital Score: 0.1197
All Score: 0.0846

Support Results
Weighted Precision: 0.7167
Weighted Recall: 0.6935

Table 4.12: Generation Experiment *giaant_t5xl_b*

The base T5 XL model performed slightly worse than T5 Large on nugget metrics, with even lower coverage of vital and all nuggets. Support metrics remained decent, showing that the model's answers were often backed by sources, but the content was less comprehensive.

- **giaant_zeph_b**: Base **Zephyr**

Nugget Results
Strict Vital Score: 0.4296
Strict All Score: 0.4177
Vital Score: 0.5311
All Score: 0.5147

Support Results
Weighted Precision: 0.6288
Weighted Recall: 0.4560

Table 4.13: Generation Experiment *giaant_zeph_b*

The base Zephyr model outperformed both T5 variants on all nugget metrics, showing much better coverage of both vital and all nuggets. Support

metrics were lower than T5 Large but still reasonable, indicating a better balance between informativeness and source support.

4.2.2 Fine-Tuned LLMs

The fine-tuning process described in previous chapters on the local LLMs using the *WebGPT* dataset was applied, with each model being fine-tuned for approximately 10 epochs, continuing until the loss stabilized. The following figures show the resulting loss values during the fine-tuning process for the **Zephyr** model (we only show first 5 epochs) and the F1 score values at the end of each epoch.

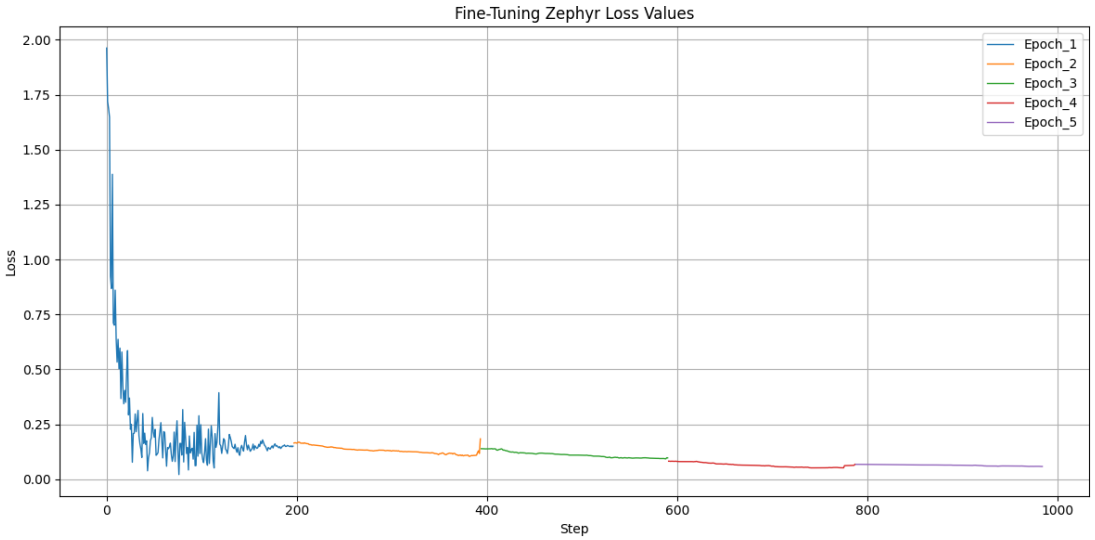

Figure 4.1: Zephyr Fine-Tuning Loss Values over first 5 epochs

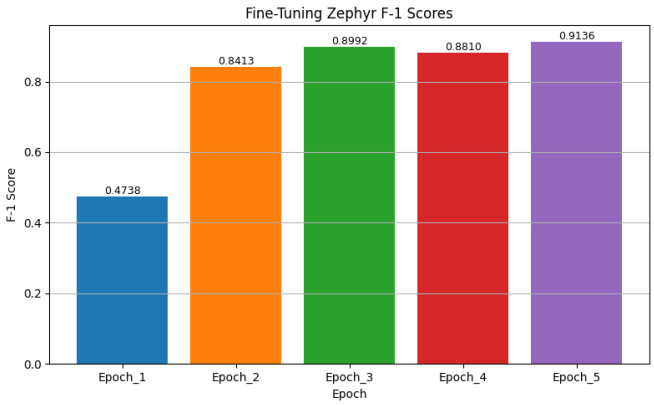

Figure 4.2: Zephyr Fine-Tuning F1 Scores at the end of each of the first 5 epochs

We here list the results of the fine-tuned models runs:

- **giaant_t5l_f**: Fine-Tuned **T5 Large**

Nugget Results
Strict Vital Score: 0.5074
Strict All Score: 0.4119
Vital Score: 0.6054
All Score: 0.4906

Support Results
Weighted Precision: 0.5580
Weighted Recall: 0.3990

Table 4.14: Generation Experiment *giaant_t5l_f*

Fine-tuning T5 Large led to a substantial improvement in nugget metrics, especially for vital nuggets, indicating that the model was able to generate more complete and relevant answers. Support metrics decreased compared to the base model, suggesting a trade-off between informativeness and strict citation support.

- **giaant_t5xl_f**: Fine-Tuned **T5 XL**

Nugget Results
Strict Vital Score: 0.4947
Strict All Score: 0.3826
Vital Score: 0.5764
All Score: 0.4650

Support Results
Weighted Precision: 0.5370
Weighted Recall: 0.4047

Table 4.15: Generation Experiment *giaant_t5xl_f*

Fine-tuning T5 XL also improved nugget metrics compared to its base version, but the scores remained slightly lower than fine-tuned T5 Large. Support metrics were similar, confirming the same trade-off.

- **giaant_zephyr_f**: Fine-Tuned **Zephyr**

Nugget Results
Strict Vital Score: 0.4875
Strict All Score: 0.4486
Vital Score: 0.5830
All Score: 0.5374

Support Results
Weighted Precision: 0.0
Weighted Recall: 0.0

Table 4.16: Generation Experiment *giaant_zephyr_f*

Fine-tuning Zephyr maintained high nugget scores, confirming its strong ability to cover relevant information. However, support metrics were zero,

indicating that the generated answers did not include or properly cite supporting sources.

4.2.3 PRAG Approach

PRAG approach was also tested but only with the **Zephyr** model since it was the only one that can handle the computational load of this approach in a reasonable time (and also in terms of required memory). Both in-context prompting and without in-context prompting approaches were tested as suggested in the original paper.

- **giaant_zpeh_prag_nc**: PRAG approach with Zephyr without in-context prompting

Nugget Results
Strict Vital Score: 0.2645
Strict All Score: 0.2053
Vital Score: 0.3628
All Score: 0.2918

Support Results
Weighted Precision: 0.0
Weighted Recall: 0.0

Table 4.17: Generation Experiment *giaant_zpeh_prag_nc*

The PRAG approach without in-context prompting resulted in moderate nugget scores, lower than fine-tuned Zephyr, and no support for citations: this is to be expected since the model was not provided with any context to generate answers from, thus it could not cite any source.

- **giaant_zpeh_prag_c**: PRAG approach with Zephyr with in-context prompting

Nugget Results
Strict Vital Score: 0.5461
Strict All Score: 0.4559
Vital Score: 0.6583
All Score: 0.5541

Support Results
Weighted Precision: 0.4750
Weighted Recall: 0.3455

Table 4.18: Generation Experiment *giaant_zpeh_prag_c*

The PRAG approach with in-context prompting achieved the highest nugget scores among all experiments, indicating excellent coverage of both vital and all nuggets. Support metrics were also reasonable, showing

that the model was able to generate informative and well-supported answers when combining PRAG with in-context prompting.

4.3 Results Comparisons

In this section, we present a comparison between the results obtained in our experiments and those of other participants in the TREC RAG 2024 Track. The goal is to contextualize the effectiveness of the proposed approaches, both in the R and RAG tasks, with respect to the SOTA and to highlight strengths and possible areas for improvement.

4.3.1 Retrieval Comparisons

We here compare the retrieval results of our runs with those of other partakers of the task: the comparison is limited to systems that used sparse indexes, excluding dense retrieval approaches to ensure fairness. The evaluation is based on the official `trec_eval` metrics, focusing on the following metrics: **MAP**, **Rprec**, precision at various cutoffs (**P@5**, **P@10**, **P@15**, **P@20**, **P@30** and **P@100**) and **Recall**, which is not directly present in the *trec_eval* output but can be derived from the `num_rel_ret` and `num_rel` values by computing the ratio of relevant documents retrieved to the total number of relevant documents for each query.

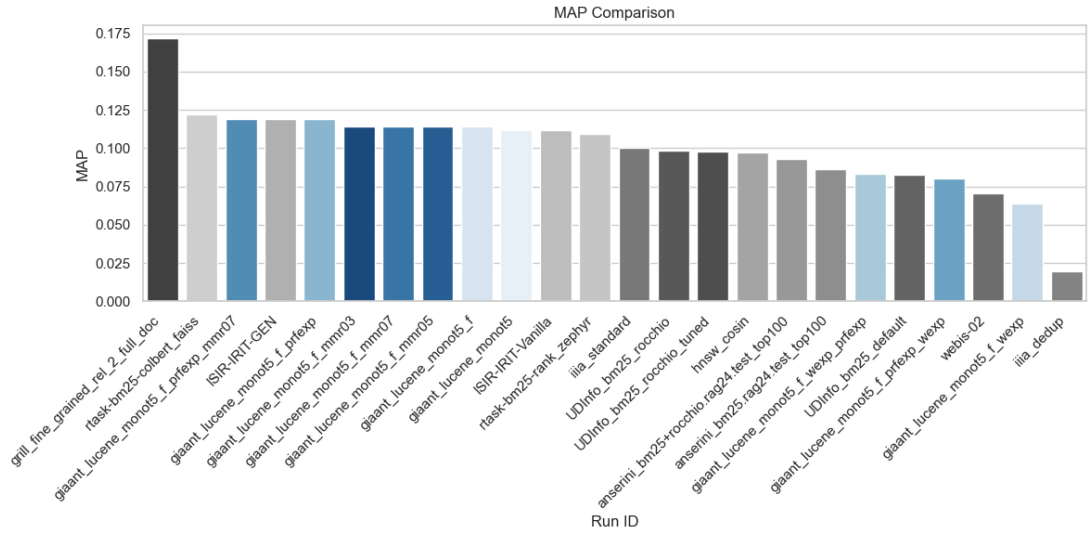

Figure 4.3: Retrieval MAP Comparisons

In Figure 4.3, it is evident that the runs using **MMR** re-ranking achieve the best MAP values among the proposed system's runs and also the majority of other participants, while synonym-based query expansion consistently leads to a decrease in performance, making the results worse than other runs and of the baseline too. **PRF**-based expansion, especially when combined with **MMR** ($\lambda = 0.7$), yields the highest MAP overall. However, in general, the differences between all sparse retrieval runs (including those of other parttakers of the task) are generally small, confirming that it is challenging to significantly outperform the baseline with sparse indexes alone.

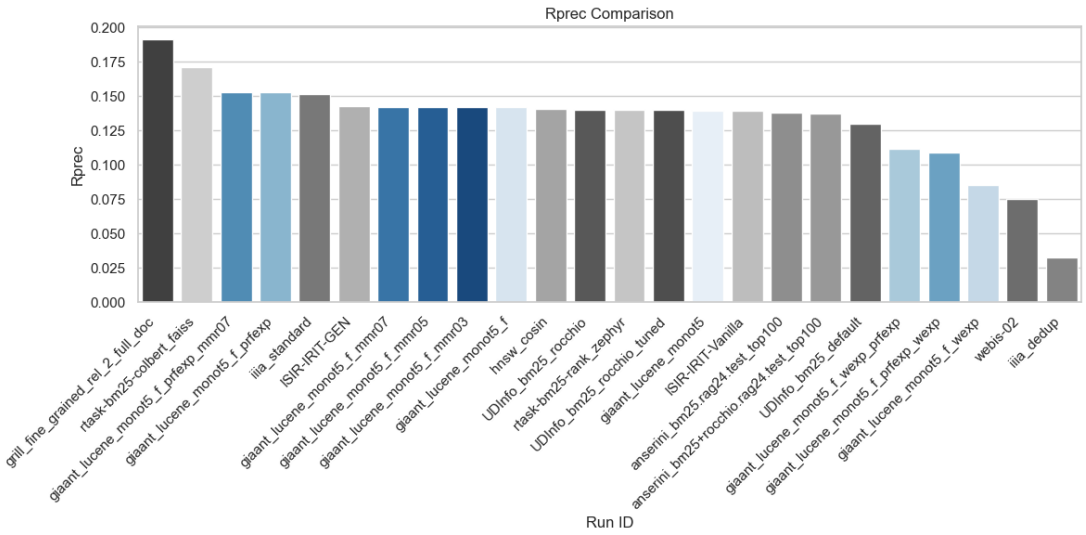

Figure 4.4: Retrieval Rprec Comparisons

The Rprec results in Figure 4.4 closely mirror the MAP trends and the overall differences between systems remain limited. One particular observation, even if not very noticeable, is that the proposed system's baseline runs (so experiments *giaant_lucene_monot5* and *giaant_lucene_monot5_f*) performed slightly worse in terms of this metric compared to MAP.

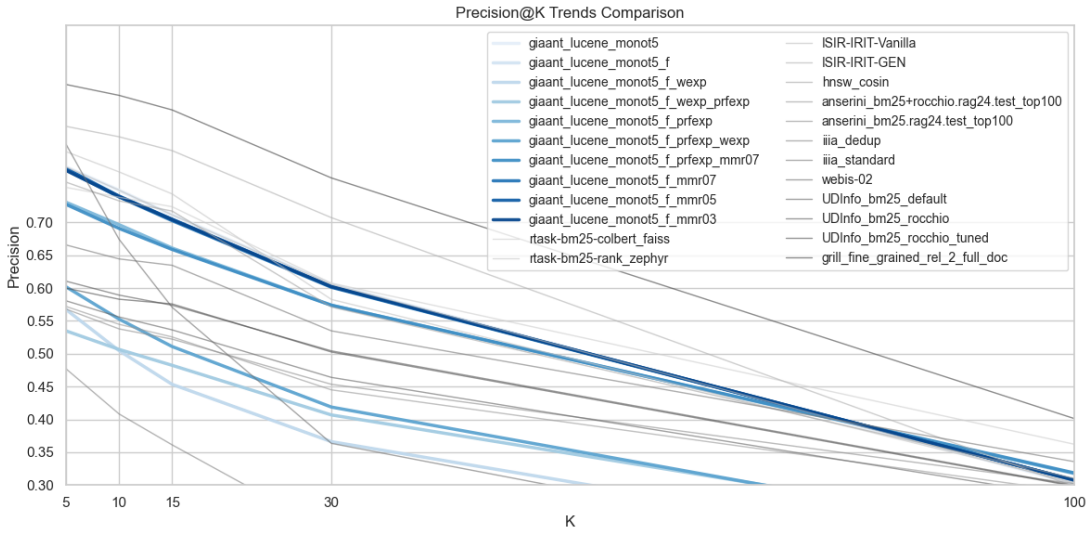

Figure 4.5: Retrieval Precision Comparisons

Figure 4.5 shows that precision drops sharply as the number of retrieved documents increases (especially after first 15 documents), which is expected given the low number of average relevant documents per query in MS MARCO dataset. The best-performing run in terms of early precision (**P@5**, **P@10**, **P@20**) is the one using only **MMR** ($\lambda = 0.5$) without PRF, justifying its selection as the main baseline. The precision trends of my runs are very similar to those of other participants.

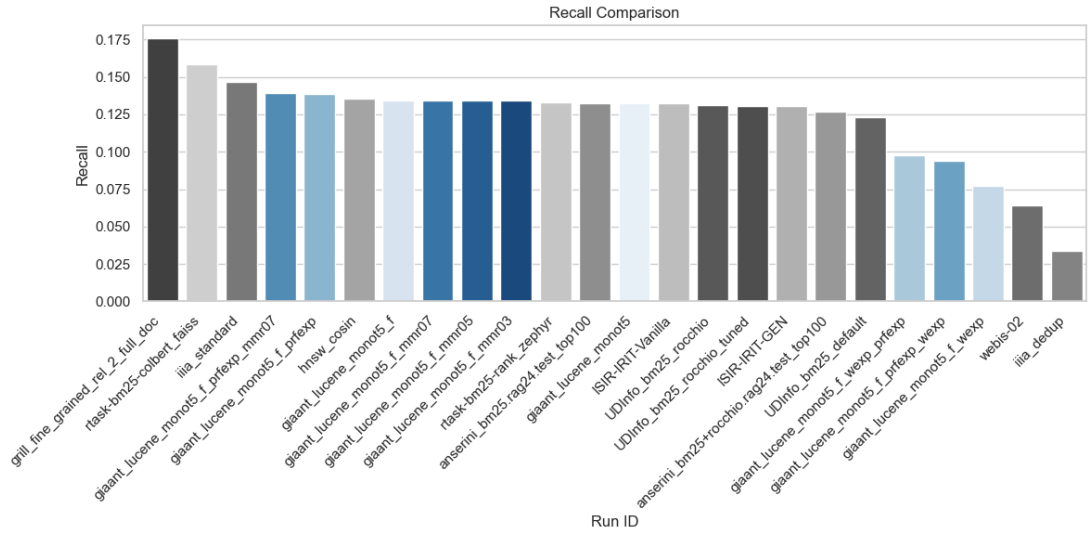

Figure 4.6: Retrieval Recall Comparisons

Figure 4.6 shows the computed Recall results, highlighting the same trends

as in the previous figures, as the metric just explains that the number of relevant documents retrieved is a not very large fraction of the total number of relevant documents for each query.

To further investigate the statistical significance of the differences between the proposed system and other participants, we performed ANOVA analyses on the main retrieval metrics. For precision, the mean of all P@k values was used. The following figures show the results for MAP, Rprec, Precision, and Recall.

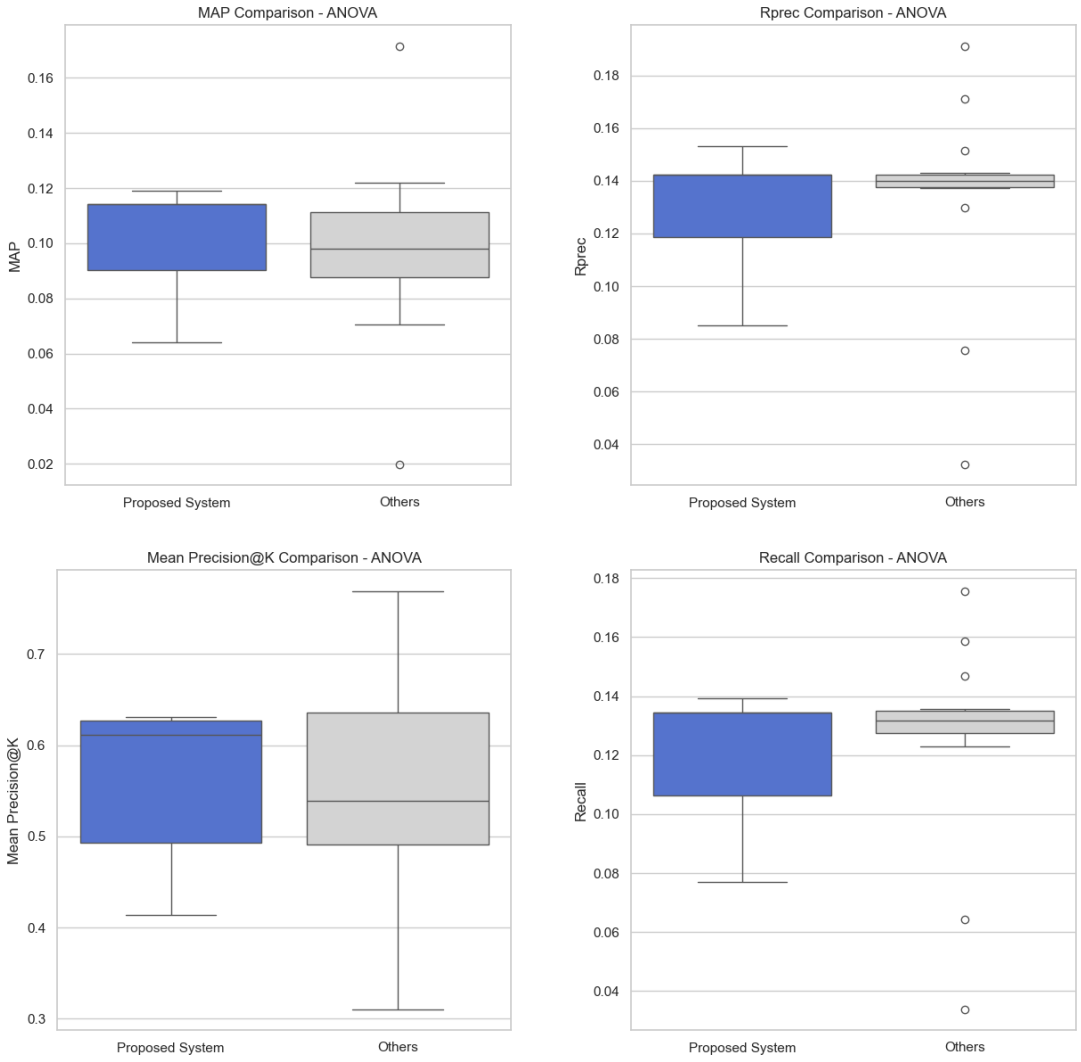

Figure 4.7: ANOVA Statistical Analysis of Retrieval Results: MAP, Rprec, Precision (mean P@k), and Recall

The ANOVA results in Figure 4.7 confirm that the proposed system's retrieval runs are statistically not significantly different from the other participants' runs, due to the utilization of sparse indexes and the nature of the

MS MARCO dataset, which makes it difficult to achieve significant improvements over the baseline. We can see that the only noticeable difference is on MAP metrics of the proposed system's runs, which are generally better than those of other participants.

4.3.2 RAG Comparisons

Finally, we compare the results of RAG experiments:

- The comparison for the **Nugget Assignment** evaluation is official [26] and directly comparable, as all participants runs got evaluated using the same evaluation tool (the *AutoNuggetizer* framework).
- For the **Support Assessment** [27], since the organizers did not provide the evaluation code, we computed the metrics by running the custom script cited in previous chapters on the downloaded runs of other participants, using the same evaluation procedure as for our results.
- **Fluency Assessment** comparisons are not reported, as neither results of other participants nor a script/guideline on how to compute those metrics are available.

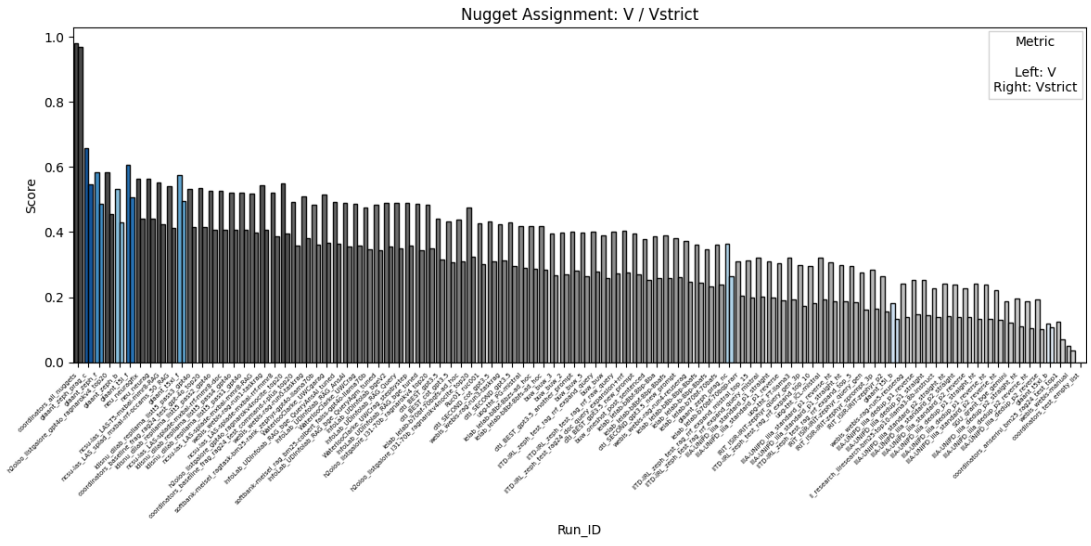

Figure 4.8: RAG Vital Nugget Comparisons

As shown in Figure 4.8, **Zephyr** achieves the best results on vital nugget metrics, especially when using the **PRAG** approach with in-context prompting, also

in respect to other participants. Fine-tuned **T5 Large** also performs very well, while **T5 XL** is generally slightly worse. The base versions of those two model score instead lower than other runs, with the second one being near the bottom of the ranking. The gap between strict and non-strict vital scores is consistent across all runs, including those of other parttakers.

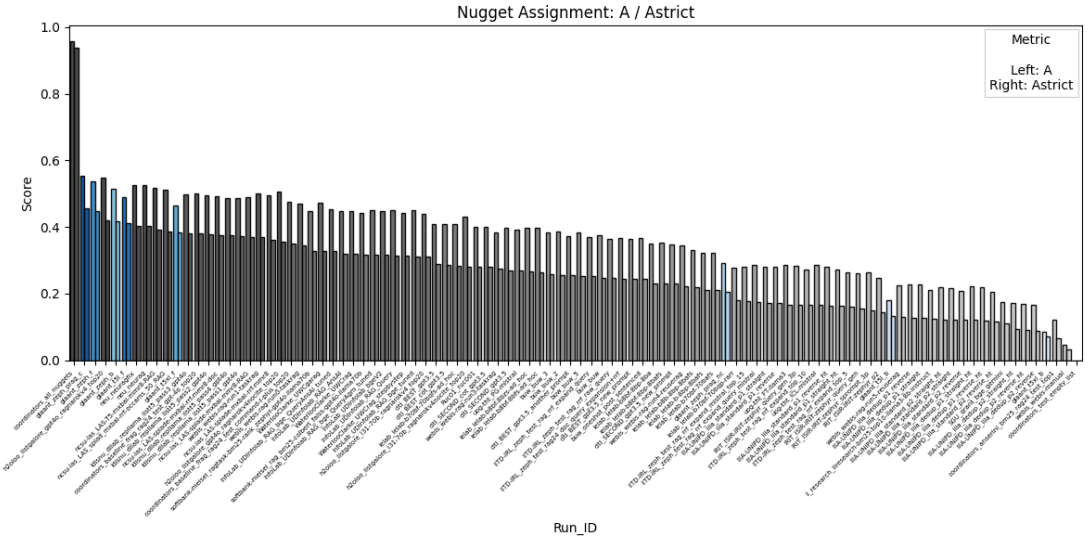

Figure 4.9: RAG All Nugget Comparisons

The trends for all nugget metrics in Figure 4.9 are very similar to those for vital nuggets, confirming the relative ranking of models and approaches. In both metrics, we can see that the **PRAG** approach without in-context prompting performs a bit worse than the average, which is actually a good result since this kind of task would not be its main purpose.

Figure 4.10 reveals that, unexpectedly, the base versions of the models often achieve higher weighted precision than their fine-tuned counterparts: this could be due to the nature of the fine-tuning process, that was probably not the best suited for this task, or to the fact that the fine-tuned models were trained on a dataset that did not align well with the evaluation criteria of this task. The only run that performs competitively with other participants is the base **T5 Large**, likely due to its inherent model characteristics, while fine-tuned **Zephyr** and Zephyr with **PRAG** without in-context prompting perform poorly, the latter as expected since it cannot cite sources. Strangely, **PRAG** with in-context prompting does not excel in this metric.

Weighted recall results in Figure 4.11 follow the same pattern as weighted precision, with the bars of the proposed system's runs being more shifted to the

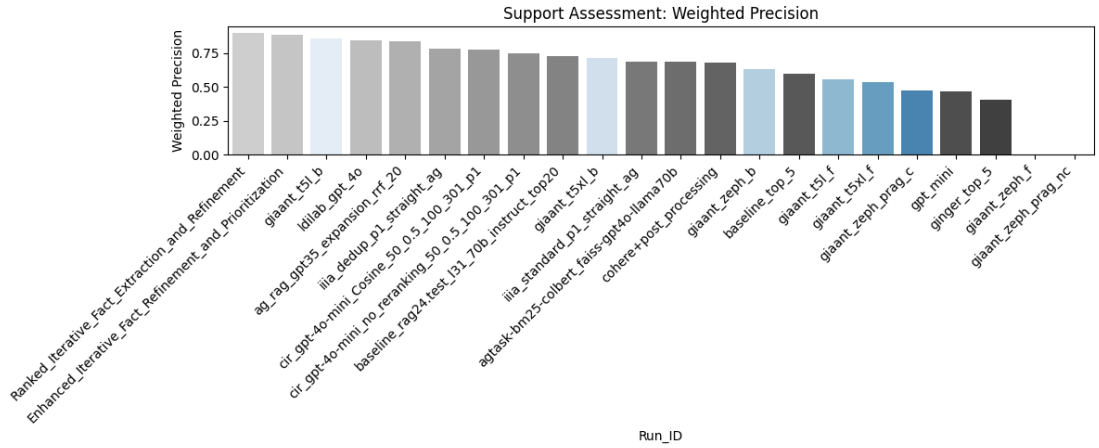

Figure 4.10: RAG Weighted Precision Comparisons

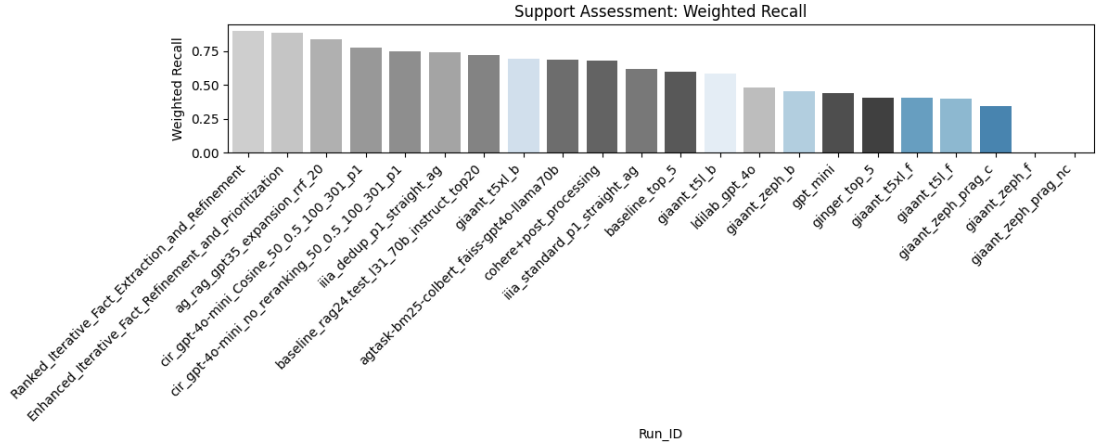

Figure 4.11: RAG Weighted Recall Comparisons

right: overall, these runs underperform compared to other participants in this metric as well. In general, experiments with models that were not fine-tuned that are better than other parttakers' runs probably achieve better results because of the optimal prompt chosen, which was tested and tuned to make sure that the model would understand how to cite sources properly.

Similarly, ANOVA analyses were performed on the main generation metrics to assess the statistical differences between the proposed system and other participants. The following figures show the results for vital and all nugget metrics (strict and non-strict), as well as weighted precision and recall.

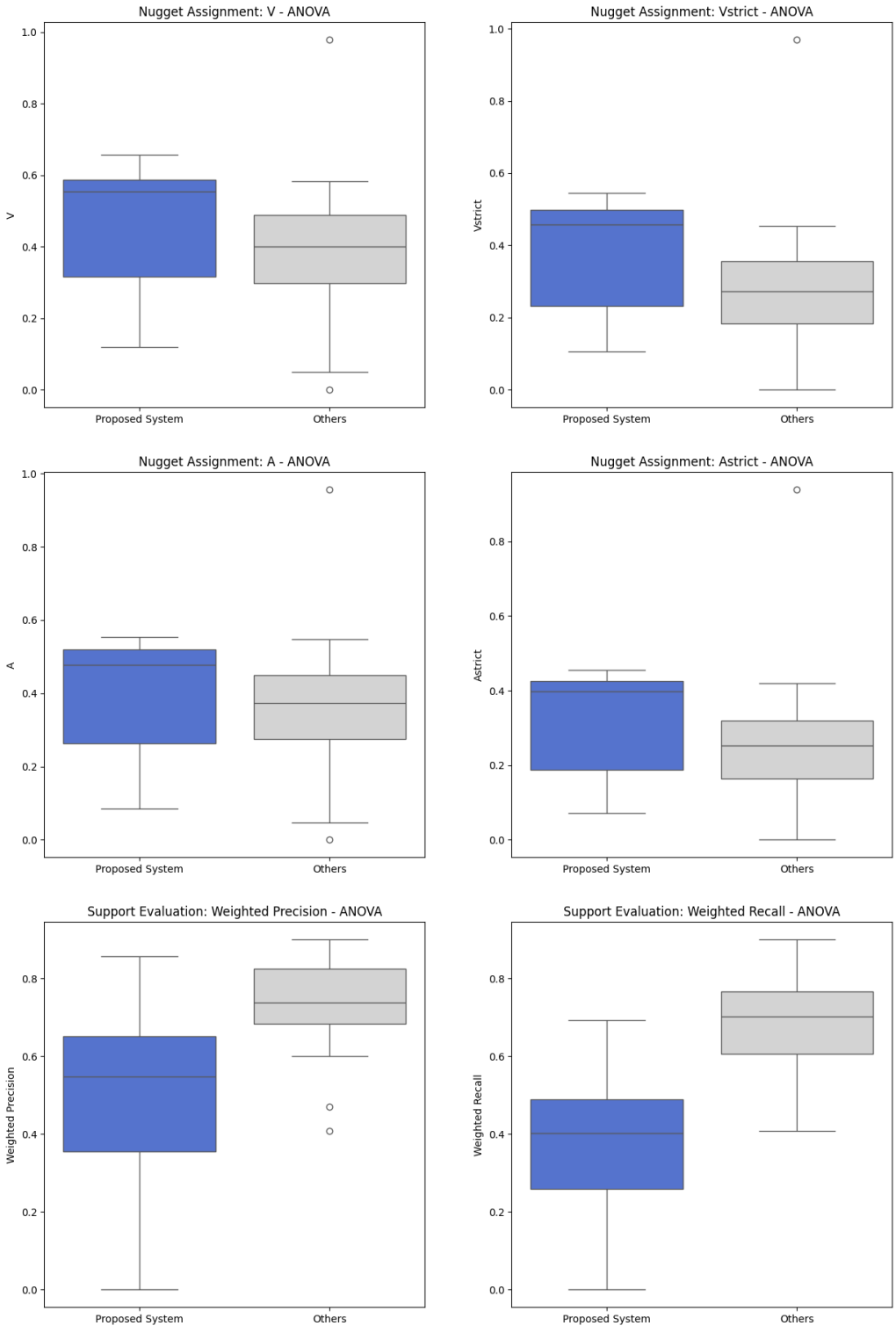

Figure 4.12: ANOVA Statistical Analysis of Generation Results: Vital/All Nugget (strict and non-strict), Weighted Precision, and Weighted Recall

The statistical analysis in Figure 4.12 highlights the challenges in achieving

consistent results in generation tasks. Notably, while nugget metrics of the proposed system's runs tend to be overall better (sometimes significantly outperforming other participants' runs), support metrics often lag behind by an important margin, reflecting the inherent difficulties in fine-tuning models for diverse generation objectives.

5

Conclusions and Future Work

The work presented in this thesis focused on the design, implementation, and evaluation of a RAG system, developed and tested in the context of the TREC RAG 2024 Track, a challenge aimed at experiment in the field of RAG for Open-Domain QA. The main objective was to build a flexible and modular architecture for this very broad task, following the guidelines and requirements of the track, with particular attention to the integration of advanced retrieval and generation techniques. The system was designed to support a variety of retrieval strategies, including different dataset choices, reranking methods such as MMR, and query expansion techniques like PRF and synonym expansion. The generation component was made highly configurable, supporting both offline and online LLMs, various prompt configurations, and fine-tuning approaches, thus ensuring extensibility and scalability. A novel paradigm, PRAG, was also explored, introducing a new way to inject knowledge into LLMs without relying on in-context prompting, which proved to be a promising direction for future research. The system was thoroughly evaluated, where possible, according to the TREC track requirements, with experiments covering multiple retrieval runs using different configurations, as well as RAG runs that combined custom retrieval results with diverse generation settings. Evaluation was conducted using both official TREC tools and custom implementations that adhered to the same metrics and principles, ensuring the reliability of the results and, overall, the system achieved strong performance in both retrieval and RAG tasks, with some runs being highly competitive. In particular, reranking with MMR and query expansion with PRF were found to be very effective for retrieval, while for

generation, models such as Zephyr and T5 Large, even in their base versions, delivered the best results for the task that required answer generation with source citation. The PRAG approach demonstrated significant potential, enabling the injection of prior knowledge into LLMs without in-context prompting, which not only accelerated the generation process but also allowed the use of smaller models; combining this new approach with in-context prompting (as proposed in the original paper) further improved effectiveness. Nevertheless, several limitations were encountered, mainly related to experimental constraints such as time and computational resources: for example, while dense indexes are preferable for retrieval due to their superior performance, they are resource-intensive and require substantial storage and compute power, making them challenging to use at scale, thus they were not tested officially in this thesis' work, which instead focused on only sparse indexes. Similarly, the proposed fine-tuning process for generation could be extended to datasets more closely aligned with RAG tasks, and additional models could be tested to further validate the findings. The PRAG approach, while promising, was only tested with Zephyr due to resource constraints, and its offline document parametrization step proved to be computationally and storage intensive (with each document requiring 5-10MB), limiting experiments to a subset of the dataset; scaling this approach to larger datasets or applying it to specific domains where such investment is feasible would be an interesting direction for future work. In conclusion, the RAG system developed in this thesis provides a solid baseline for further experimentation and extension in the field of RAG for Open-Domain QA. The results obtained demonstrate the strengths of the proposed architecture and the potential of novel paradigms being tested in these environments, while also highlighting clear opportunities for future research and improvement—particularly in optimizing resource usage, expanding experimental coverage, and exploring new model and retrieval configurations. Building upon this baseline will enable more extensive testing and development, contributing to the ongoing advancement of RAG systems and their applications.

References

[1] Stephen Robertson and Hugo Zaragoza. "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Foundations and Trends in Information Retrieval* 3 (Jan. 2009), pp. 333–389. DOI: 10.1561/1500000019.

[2] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023. arXiv: 1910.10683 [cs.LG]. URL: https://arxiv.org/abs/1910.10683.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[4] Jaime Carbonell and Jade Stewart. "The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries". In: *SIGIR Forum (ACM Special Interest Group on Information Retrieval)* (June 1999). DOI: 10.1145/290941.291025.

[5] Hang Li, Ahmed Mourad, Shengyao Zhuang, Bevan Koopman, and Guido Zuccon. *Pseudo Relevance Feedback with Deep Language Models and Dense Retrievers: Successes and Pitfalls*. 2022. arXiv: 2108.11044 [cs.IR]. URL: https://arxiv.org/abs/2108.11044.

[6] Peilin Yang, Hui Fang, and Jimmy Lin. "Anserini: Enabling the Use of Lucene for Information Retrieval Research". In: New York, NY, USA: Association for Computing Machinery, 2017. ISBN: 9781450350228. DOI: 10.1145/3077136.3080721. URL: https://doi.org/10.1145/3077136.3080721.

[7] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. *Pyserini: An Easy-to-Use Python Toolkit to Support Replicable IR Research with Sparse and Dense Representations*. 2021. arXiv: 2102.10073 [cs.IR]. URL: https://arxiv.org/abs/2102.10073.

[8] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. *The Faiss library*. 2025. arXiv: 2401.08281 [cs.LG]. URL: https://arxiv.org/abs/2401.08281.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[10] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[11] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG]. URL: https://arxiv.org/abs/1711.05101.

[12] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG]. URL: https://arxiv.org/abs/1912.01703.

[14] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. *8-bit Optimizers via Block-wise Quantization*. 2022. arXiv: 2110.02861 [cs.LG]. URL: https://arxiv.org/abs/2110.02861.

[15] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. *MS MARCO: A Human Generated MAchine Reading COmprehension Dataset*. 2018. arXiv: 1611.09268 [cs.CL]. URL: https://arxiv.org/abs/1611.09268.

[16] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013/.

[17] Ronak Pradeep, Nandan Thakur, Shivani Upadhyay, Daniel Campos, Nick Craswell, and Jimmy Lin. *The Great Nugget Recall: Automating Fact Extraction and RAG Evaluation with Large Language Models*. 2025. arXiv: 2504.15068 [cs.IR]. URL: https://arxiv.org/abs/2504.15068.

[18] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Tim Rocktäschel, and Sebastian Riedel. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2020. arXiv: 2005.11401 [cs.CL]. URL: https://arxiv.org/abs/2005.11401.

[19] Weihang Su, Yichen Tang, Qingyao Ai, Junxi Yan, Changyue Wang, Hongning Wang, Ziyi Ye, Yujia Zhou, and Yiqun Liu. *Parametric Retrieval Augmented Generation*. 2025. arXiv: 2501.15915 [cs.CL]. URL: https://arxiv.org/abs/2501.15915.

[20] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL]. URL: https://arxiv.org/abs/2302.13971.

[21] Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. *Zephyr: Direct Distillation of LM Alignment*. 2023. arXiv: 2310.16944 [cs.LG]. URL: https://arxiv.org/abs/2310.16944.

[22] Edward Loper and Steven Bird. *NLTK: The Natural Language Toolkit*. 2002. arXiv: cs/0205028 [cs.CL]. URL: https://arxiv.org/abs/cs/0205028.

[23] Wes Mckinney. "pandas: a Foundational Python Library for Data Analysis and Statistics". In: *Python High Performance Science Computer* (Jan. 2011).

[24] George A. Miller. "WordNet: a lexical database for English". In: 38.11 (1995). ISSN: 0001-0782. DOI: 10.1145/219717.219748. URL: https://doi.org/10.1145/219717.219748.

[25] Ronak Pradeep, Nandan Thakur, Sahel Sharifymoghaddam, Eric Zhang, Ryan Nguyen, Daniel Campos, Nick Craswell, and Jimmy Lin. *Ragnarök: A Reusable RAG Framework and Baselines for TREC 2024 Retrieval-Augmented Generation Track*. 2024. arXiv: 2406.16828 [cs.IR]. URL: https://arxiv.org/abs/2406.16828.

[26] Ronak Pradeep, Nandan Thakur, Shivani Upadhyay, Daniel Campos, Nick Craswell, and Jimmy Lin. *Initial Nugget Evaluation Results for the TREC 2024 RAG Track with the AutoNuggetizer Framework*. 2024. arXiv: 2411.09607 [cs.IR]. URL: https://arxiv.org/abs/2411.09607.

[27] Nandan Thakur, Ronak Pradeep, Shivani Upadhyay, Daniel Campos, Nick Craswell, and Jimmy Lin. *Support Evaluation for the TREC 2024 RAG Track: Comparing Human versus LLM Judges*. 2025. arXiv: 2504.15205 [cs.CL]. URL: https://arxiv.org/abs/2504.15205.

Acknowledgments

Quando ho iniziato questo percorso, non sapevo esattamente dove mi avrebbe portato. Ma una cosa l'ho capita lungo la strada: nella vita non c'è niente di garantito, tranne che tutto può cambiare in un attimo. E se hai l'opportunità di giocarti questa partita, devi cercare di apprezzare ogni singolo momento. Molte persone non capiscono il valore di ciò che hanno finché non lo perdono. E poi, quando è troppo tardi, iniziano a raccontare storie del passato, "ti ricordi quando...". Ma ora è il mio momento, e voglio ringraziare la mia famiglia, Stella, i miei amici, Dio e tutti coloro che mi hanno supportato e creduto in me. Ho intenzione di festeggiare. Ho intenzione di festeggiare, urlare, stappare una bottiglia ogni volta che ne avrò l'occasione, perché ce l'ho fatta! So che tanti si chiedevano quanto ci avrei impiegato, se ne valesse la pena, cosa avrei fatto se non fossi riuscito a finire... credo che non lo scopriremo mai.