Overview of Hybrid Retrieval Methodologies in Retrieval-Augmented Generation (RAG)

Siddhartha Somani
Georgia Institute of Technology
ssomani37@gatech.edu

***Abstract*— Retrieval-Augmented Generation (RAG) is a prominent and effective method of elevating the quality and relevance of generated content by integrating Large Language Models (LLMs) with a private text corpus or knowledge base for contextualized and fact based responses. However, as document size and dataset dimensionality and density scale up, the ability to maintain the accuracy and factual consistency of a standard RAG model becomes continuously more challenging. Hybrid RAG models represent a promising advancement in alleviating this issue by integrating diverse data processing capabilities within a unified framework. This study provides a survey of prevalent algorithms used for vector and semantic search, discusses various fusion architectures for merging results, and analyzes their respective performance on gold standard Q&A (question-answering) datasets such as SQuAD and Google's NQ (Natural Questions) Dataset. This study aims to offer insights into past developments in Hybrid RAG methodologies and potential advancements of Hybrid RAG models, setting the stage for future research in integrated generative AI systems.**

1 Introduction

1.1 RAG

RAG combines relevant documents from large datasets and utilizes generative models like LLMs to produce responses that are both contextually informed and factually correct for QA (question-answering) queries [1]. In RAG, the model leverages a vast corpus of documents or a knowledge base to retrieve relevant information that augments its generative capabilities [2]. This approach is particularly effective in the following scenarios (but not limited to):

- Question and Answering [2]: ask specific questions with a resulting answer based on the provided context or knowledge within the documents
- Search Augmentation [2]: augment search results to enable efficient identification of relevant information to the user from the knowledge base
- Knowledge Engine [1, 2]: combines LLMs with a retrieval system to access provided information and generate more accurate and contextually relevant responses

By combining retrieval with generation, RAG bridges the gap between purely generative models, which may struggle with recall accuracy or specificity, and retrieval-based models, which may lack the fluidity and creativity of generative models. The dynamic interaction between retrieval and generation allows RAG models to produce more informed and contextually relevant outputs, making them ideal for applications like question answering, summarizing, and content generation.

1.2 RAG Architecture: Retriever & Generator

Standard RAG architecture consists of multiple crucial components that exemplify its retrieval and generative abilities on provided information, and are constructed to integrate those components seamlessly. The effectiveness of RAG architecture typically depends on two main elements: a retriever and a generator [3]. The retriever identifies and selects the top-k documents, or relevant information, from the provided corpus, which are then fed into the generator. The generator, often embodied as a LLM or as an open-source transformer model, utilizes and condenses the retrieved information to produce contextually enriched responses [1].

1.3 Sparse vs. Dense Retrieval

There are two types of retrieval in NLP that are used in RAG: (1) Sparse Retrieval and (2) Dense Retrieval [4]. *Sparse Retrieval* is a method of information retrieval that relies on keyword matching and term-frequency statistics to identify relevant documents from a large corpus. It focuses on analyzing the frequency and distribution of each term within the query located within the provided documents, often using techniques such as term frequency (TF) or inverse document frequency (IDF) [3]. Sparse retrieval methods are designed to handle high-dimensional data efficiently by representing documents and queries as sparse vectors, where most elements are zero [4]. This approach excels in scenarios where the textual content is critical, and the goal is to quickly locate documents that contain specific keywords or phrases relevant to the query. *Dense Retrieval*, on the other hand, maps the query to vector representations of the provided text to find relevant documents. Unlike Sparse Retrieval, which relies on keyword matching, Dense Retrieval encodes both queries and documents into dense, high-dimensional vectors. This allows for more nuanced and semantic matching, as it captures contextual meanings and relationships between terms [4]. Dense retrieval methods excel in scenarios that require a deep understanding of text meaning and semantics, enabling the retrieval of documents based on their overall content and relevance to the query rather than exact keyword matches [5].

1.4 Hybrid RAG

As document sizing and query complexity scale, standard RAG models that implement either sparse or Dense Retrieval

become more and more inaccurate. Therefore, the following question is introduced: can a fusion between the two return better results? The answer is a resounding, yes [3]. The utility and efficacy of a hybrid model between both sparse and dense search is known as Hybrid RAG. This technique and revised model structure implements a consolidated retrieval method that combines both sparse and dense Retrieval algorithms and filters the top results between the two [3]. Hybrid Retrieval combines the strengths of both sparse [4] and dens [5] retrieval methods to enhance information retrieval performance. By integrating keyword-based approaches with semantic vector representations, hybrid retrieval can effectively capture both exact term matches and deeper contextual relationships [3]. This dual strategy allows for more comprehensive retrieval by addressing the limitations of each method alone — sparse retrieval's reliance on exact matches and dense retrieval's computational demands. The combination results in a more robust system that can efficiently and accurately identify relevant documents across a wide range of queries, providing a balanced solution that leverages the advantages of both retrieval paradigms [2–4, 6].

1.5 Focus of Study

This study focuses primarily on delving into RAG, current retrieval mechanisms, and the power of hybrid retrieval models. This study does a high level elaboration of standard RAG architecture and common technologies used in modern day generative AI development [1]. Furthermore, this study covers the various types of sparse algorithms, like TF-IDF or BM25 [7], and dense search techniques like embeddings and transformer models, neural networks, and k-nearest neighbors. This study then goes on to provide empirical findings of evaluating a baseline hybrid model on two prominent datasets for NLP and QA, which are SQuAD [8] and Google's NQ Dataset [9]. The primary goal of the study is to demonstrate techniques that can be utilized to improve the baseline performance of smaller scale transformer models QA pipeline models and sentence transformer models that include a fraction of the number of parameters of LLMs, ranging in the hundred millions instead of the billions. Due to LLMs being extremely computationally expensive to run from a local device, smaller models need to be improved gradually to provide complementary services to these larger LMs, and therefore this study focuses on one key approach, hybrid retrieval, and elaborating on how it results in improvement from baseline RAG models using standard retrieval architecture.

2 Related Work

Retrieval has been a primary area of RAG research, as methods to improve efficiency, accuracy, and costliness often stem from enhancing retrieval capabilities of a model. Hybrid search contains two primary components, sparse retrieval and dense retrieval, and as such, each have their own separate lines of research.

2.1 Sparse Retrieval

The core of sparse retrieval originate from traditional concepts like Term Frequency (TF) and Inverse Document Frequency (IDF). Conventional algorithms like TF-IDF (Term Frequency - Inverse Document Frequency) have paved the path for many recent advancements in sparse term search within RAG. The BM25 (Best Match 25) algorithm, established as an improved method from TF-IDF, leverages the function of TF-IDF and uses similarity search to map documents based on their pertinence to the query, as investigated by Robertson and Zaragoza (2009) [7]. The study thoroughly investigates the probabilistic nature of BM25 along with its saturation function to address the short comings of TF-IDF. Similar to BM25, Okapi BM25 from TREC-3 (1992), which is a Text Retrieval Conference, [10] is a variant of the original BM25 algorithm that focuses on optimizing hyperparameters that regulate the significance of term frequency and document length of BM25, leading to higher accuracy in results and frequency scores. Another algorithm that focuses on the growing field of sparse retrieval includes SPLADE [11] and its successor SPLADE v2 [12]. The initial study for SPLADE, which stands for Sparse Lexical and Expansion Model for First Stage Ranking, demonstrated how sparse vector representations using a BERT encoding model for sparse embeddings. Unlike traditional methods such as BM25 and TF-IDF, which rely on exact term matching and fixed hyperparameters, the study shows how SPLADE learns term expansions to enhance document-matching capabilities, effectively combining the efficiency of sparse retrieval with the semantic understanding of dense models without conducting dense retrieval explicitly. SPLADE v2 further builds on its predecessor by modifying the pooling mechanism through the installation of distillation models evaluated on the BEIR benchmark [13]. SPLADE v2 was conducted during TREC DL and showed a 9% improvement on the original model.

2.2 Dense Retrieval

Key developments in dense retrieval include the use of embedding techniques such as BERT [14] and its derivatives, which improve the quality of vector representations by incorporating contextual information. Google AI's team who led the study show how BERT uses a bi-directional encoding technique that process input sequences in parallel by considering both left and right directions simultaneously instead of using unidirectional encoding. Another method for dense retrieval known as Siamese networks have been employed to learn similarities between pairs of inputs, as they employ two identical subnetworks for one-shot classification, enabling more accurate retrieval based on semantic relevance. Reimers and Gurevych (2019) illuminate how the proposed model, S-BERT (Sentence BERT) uses siamese networks coupled with BERT to enhance semantically meaningful embeddings while reducing the time taken for pair similarity identification [15]. The Dense Passage Retrieval (DPR) framework [16] further enhances retrieval efficiency by utilizing separate encoders for queries and passages, otherwise known as dual-encoding [17], which is shown by Karpukhin and Oguz (2020) to significantly improve performance over traditional sparse-based methods like BM25 [5].

2.3 Approximate Nearest Neighbors

ANN (Approximate Nearest Neighbors) Algorithms are used to finds data points that are *near* to a query or input point, but not *exactly* near to the input. RAG systems often employ ANN search instead of exact nearest neighbor search

for large-scale vector retrieval due to its efficiency. This allows RAG to quickly find relevant context from massive document collections. However, the increased efficiency of ANNs come with a corresponding trade-off of accuracy. PipeRAG [18], a study done to investigate faster methods of retrieval in RAG, demonstrates the efficacy of ANNs in search and retrieval, along with its shortcomings. The studies demonstrates how the database is partitioned based on RETRO chunking [19]. Furthermore, ANNs are used for retrieval, alongside bi-directional dual-encoding techniques coupled with multi-head attention mechanisms to avoid high computational demands. PipeRAG elucidates the trade-off of ANNs regardless of index types and hyperparameter tuning [18], proving their effectiveness in accelerated retrieval but corresponding inaccuracy in retrieved documents.

2.4 Knowledge Distillation

Recent work in distillation techniques in semantic search retrieval have emerged. The core concept behind Knowledge distillation is to train or fine-tune a smaller *student* model on a larger *teacher* model on specific tasks in order for the smaller model to replicate the behavior of the larger model. This process enables for memory computationally efficient models that can be used to perform advanced and wide ranging tasks. A recent study, D2LLM [20], investigates the decomposition of LLMs and its integration with knowledge distillation techniques to go from teacher language models (larger and more complex) to student language models (smaller and more efficient). D2LLM proposes that after applying the decomposition and distillation to the smaller method, its evaluated scores were similar to the larger model, but was producing output at a much lower latency.

2.5 Re-ranking Methods

A powerful technique in RAG that is often utilized directly post-retrieval is *re-ranking*. Re-ranking is a sophisticated technique to hyper-enhance the relevance of retrieved documents. In other words, in juxtaposition with dense or sparse retrieval, re-ranking is able to reorganize and filter the responses, or rank the retrieved results and re-iterate through the model, for improved generated output. Nogueira and Cho (2020) conducted a study which explored a re-implementation of BERT that adjusted the model for query-passage re-ranking [21]. The study revealed the empirical results of the re-implemented re-ranking based BERT which was evaluated on the TREC-CAR dataset along with the MS MARCO retrieval task. The revised BERT model was fine-tuned for the re-ranking task using the cross-entropy loss function [22]. The re-ranking BERT model outperformed the base model by 27% over 6800 queries and 1000 passages for retrieval context. This study further highlights the utility of advanced techniques that transcend the process of retrieval, and occur post-retrieval to enhance the re-iterated retrieval process while re-ranking.

2.6 Hybrid Retrieval

The salient feature of hybrid retrieval is the combination of sparse and dense retrieval methods. A recent study, BlendedRAG [3], proposes a hybrid RAG architecture that combines dense vector representations (through the use of transformers, encoding models, and K-Nearest Neighbors) with sparse term frequency and document length algorithms (e.g BM25, TF-IDF, etc.) to enhance information retrieval. By leveraging dense vector representations using advanced transformers and encoding models alongside traditional sparse algorithms like BM25 and TF-IDF, BlendedRAG achieves a more robust retrieval mechanism. This hybrid approach effectively balances the strengths of both retrieval methods: the contextual richness of dense embeddings and the precision of sparse algorithms in handling specific term frequencies. The architecture is tested across various datasets, such as SQuAD, NQ, TREC-COVID, and CoQA, demonstrating its superior performance compared to individual retrieval methods like KNN and BM25. These evaluations underscore BlendedRAG's ability to improve the accuracy and relevance of information retrieval in complex and high dimensional datasets. BlendedRAG not only provides a stellar example of how integrated retrieval techniques enhance RAG accuracy, but also how it can be applied in non-theoretical domains (e.g finance, law, medicine, etc.), paving the pathway for future developments.

3 RAG Architecture

With the rise of Large Language Models (LLMs) and their corresponding utility, many techniques have been developed to further enhance their potency and dynamism. Retrieval Augmented Generation (RAG) has proven to be extremely successful in enhancing LLM performance when paired with a pretrained or fine-tuned model [1]. RAG, in fact, has transcended as simply being a technique to interact with uploaded documents. It has transformed into a model architecture for advanced NLP tasks and content specific use of LLMs. RAG architecture is composed of several key elements, all shown in Figure 1 which models the standard architectural practice for question answer retrieval augmented generation [1]. The initial stage in QA RAG begins with the user. The first step is for the user to upload documents. These documents are then split into *chunks*, which are a partition of the document, and have a specified or dynamic number of characters depending on the document length and the preference of the creator of the RAG model. There are two parameters that exist when chunking: chunk size and overlap. Chunk size is simply the number of character, or the size, of the partition or chunk of the document. Overlap, is the number of character between each chunk that overlap from one to the other. For example, consider the sentence, "the quick brown fox jumps over the lazy dog". Without any overlap, chunking of this sentence based on nouns and verb phrases might look like this:

1. Noun Phrase: "The quick brown fox"
2. Verb Phrase: "jumps over"
3. Noun Phrase: "the lazy dog"

Now, consider a situation where there is overlap:

1. Noun Phrase: "The quick brown fox"
2. Verb Phrase: "fox jumps over"
3. Noun Phrase: "jumps over the lazy dog"

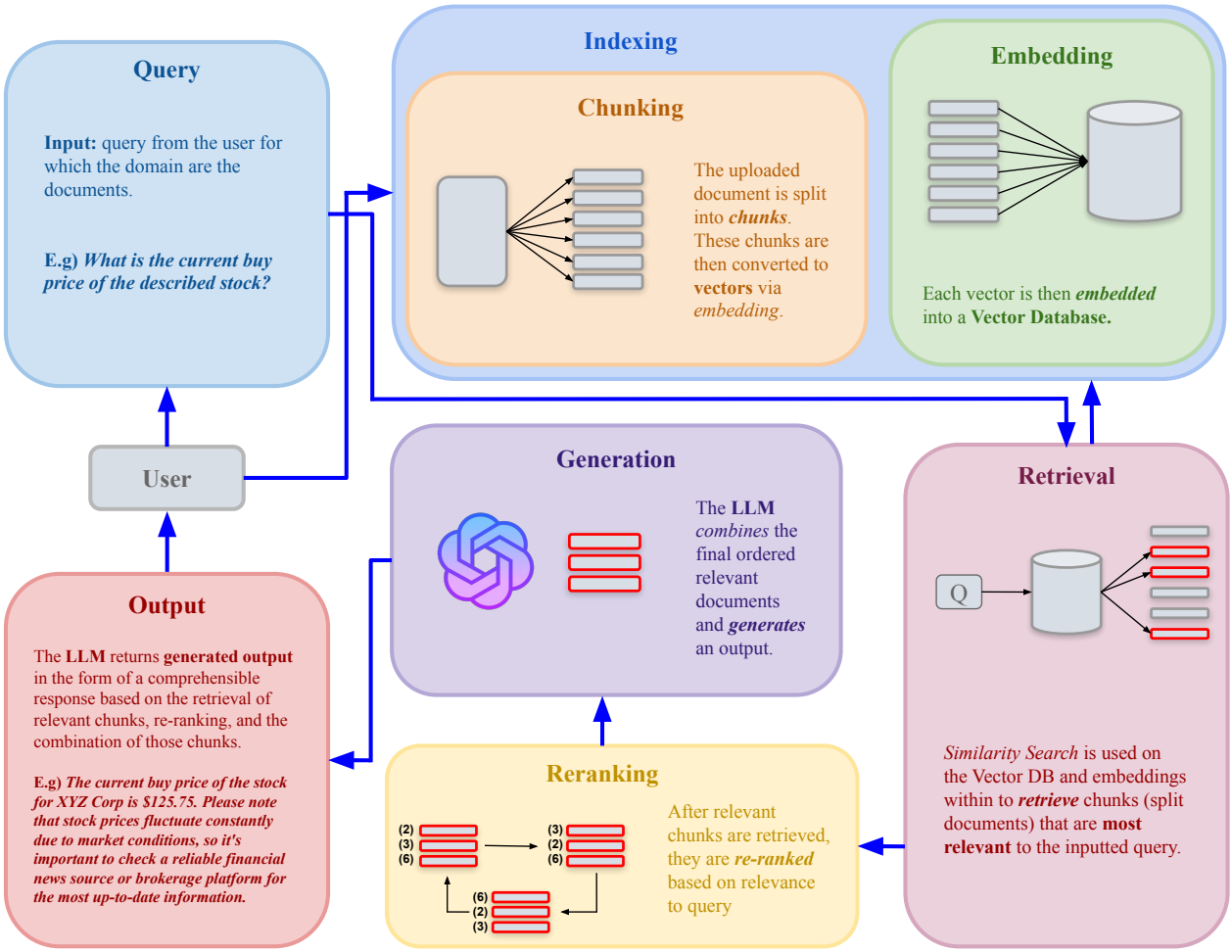

Figure 1: Representation of standard QA (Question Answering) RAG Architecture. Primary procedures include (1) Querying, (2) Indexing (chunking + embedding), (3) Retrieval, (4) Reranking, (5) Generation, and (6) Output. Example query and output are provided through the use of GPT 3.5, simply to demonstrate the usage of a generative language model or LLM for language purposes.

The overlap is clear between each chunk. The primary rationale for overlap, is so that each chunk can maintain a sense of context and semantic meaning between the surrounding chunks. A greater overlap would mean high contextual meanings are extracted, but low distinguishable features between each chunk, which is a trade off (specifically when it comes to *retrieving* the relevant chunks). The next step after chunking the document, is to encode each chunk into a *vector embedding*, which is a vector representation of the chunk. These vector embeddings are then stored in a *Vector Database (DB)*, which serves as the knowledge base for the retriever. The combination of *chunking* and *embedding* is known as *indexing* in RAG [1, 6]. Indexing refers to the process of organizing and storing information from a large corpus so that it can be quickly and efficiently retrieved to support the generation of responses, most often in the form of a Vector DB in RAG. After the document has been effectively chunked, embedded, and indexes successfully, the user then asks a query to the RAG model. The query is then encoded into its own vector representation, and compared by the *retriever*.

Retrieval is the process of conducting or performing a similarity search algorithm on the Vector DB to extract the relevant documents based on the vector representation of the query. Relevant Documents are specific vectors from the Vector DB that represent chunks or partitions of the original documents that contain information relevant to the user's query. Once the relevant documents are retrieved, a procedure of *ranking* takes place. Ranking is the process of ordering the retrieved relevant documents by most to least relevant in relation to the query. *Reranking*, is often a technique used to enhance the results of ranking and mitigate potential errors in early steps. Reranking can be done iteratively or recursively [1]. Once the final ordering of the retrieval relevant documents is achieved, these documents and their specific order is then processed in the step of *generation*. The retrieved documents are then combined with the original query to form a context-rich input for the LLM, or any generative language model. The generative model, typically based on transformer architectures like GPT, Llama, any HuggingFace pretrained LMs, or similar, uses the combined input (query + retrieved information) to generate a coherent and contextually appropriate response. The model leverages its semantic and language understanding capabilities, along with the retrieved context, to produce outputs that are both fluent and relevant. A study conducted in Fudan University in

Shanghai, China (2024) explores the best practices and technologies that are used in modern RAG [6]. Figure 1 illustrates the standard architecture of RAG models, but many studies have investigated more advanced classifications of RAG. Specifically, [1] goes beyond the standard practice for RAG and compares the three paradigms of RAG: Naive RAG, Advanced RAG, and Modular RAG. This study is focused towards the retrieval portion of RAG and therefore maintains the use of the standard RAG model to isolate improvements to be a result of enhanced retriever configurations.

4 Sparse Retrieval

4.1 Bag-of-Words (BoW)

There exists a plethora of techniques to conduct effective and accurate sparse retrieval for an inputted query over a text corpus. One of the core and fundamental algorithms or techniques used in sparse retrieval is the Bag-of-Words (BoW) model. The BoW model represents textual data as an *unordered* collection of words, disregarding grammar and word order, and focuses solely on term frequency. This approach results in a sparse vector for each document within the corpus (each entry of the vector is a mapping of the frequency for each distinct term). This method results in a sparse representation of the corpus where many positions in the vector are zero, corresponding to terms not present in a given document within the larger corpus.

Let $V = \{w_1, w_2, ...w_n\}$ represent the set of all unique words within the corpus. Each document d can be represented as the vector $v_d = f(w_1, d), f(w_2, d), ...f(w_n, d)$, where $f(w_i, d)$ represents the raw frequency of the term w_i in the document d as

$$\mathrm{f}(w, d) = \sum_{w_i \in d} f_{w_i, d}$$

Each document is represented as a vector in a n-dimensional space, where dimension of v_d is equal to the cardinality of the unique terms in the corpus as

$$dim(v_d) = |V|$$

and the corpus matrix $C = [[v_{d1}], [v_{d2}], [v_{d3}], ..., [v_{dn}]]$ is modeled as

$$C = \begin{bmatrix} v_{d11} & v_{d12} & \cdots & v_{d1n} \\ v_{d21} & v_{d22} & \cdots & v_{d2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{dn1} & v_{dn2} & \cdots & v_{dnn} \end{bmatrix}$$

When using BoW for sparse retrieval, the user still needs to generate a query. Similar to the documents in the corpus, this query is encoded into a vector v_q where the dimension is the same as the dimension of the document vector v_d but the frequencies are mapped from the terms in the query. The similarity between a document vector v_d and a query vector v_q can be computed in various ways. One of the most common methods is by computing the *Cosine Similarity Search* between v_q and v_d. Cosine Similarity Search is calculated as the cosine of the angle between them. This measure is particularly efficient for sparse vectors, as computed in the BoW model, because it only requires computation on the *non-zero* dimensions, making it computationally efficient for high-dimensional and sparse data sets. Cosine Similarity Search is calculated as

$$\text{Cosine Similarity}(v_q, v_d) = \frac{v_q \cdot v_d}{\|v_q\| \|v_d\|}$$

This value enables the retriever to find all document vectors that are most "similar" or dimensionally close to the query vector.

4.2 TF-IDF

An enhancement and further development of the Bag-of-Words model is TF-IDF (Term Frequency - Inverse Document Frequency) [3]. TF-IDF is a known statistical measure used in modern NLP to evaluate the importance of a word in a document relative to a collection of documents, or a corpus. TF-IDF value increases proportionally with the number of times a word appears in the document but is offset by the frequency of the word in the corpus. TF-IDF is composed of two components: Term Frequency (TF) and Inverse Document Frequency (IDF). Term Frequency measures the *frequency* of a term in a document. Term Frequency is calculated as

$$\mathrm{TF}(t, d) = \sum_{t' \in d} f_{t', d}$$

where $f_{t,d}$ is the raw frequency of term t in document d. Inverse Document Frequency measures the *importance* of a term in the corpus.

$$\mathrm{IDF}(t, D) = \log \frac{|\{d \in D : t \in d\}|}{|D|}$$

where $|D|$ is the total number of documents, and $|\{d \in D : t \in d\}|$ is the number of documents containing the term t. TF-IDF combines TF and IDF and is the resulting product, as such:

$$\text{TF-IDF}(t, d, D) = \mathrm{TF}(t, d) \cdot \mathrm{IDF}(t, D)$$

4.3 BM25

TF-IDF, while being a core algorithm for sparse retrieval and an improvement over the BoW model, is often deferred to the use of even more enhanced and improved term frequency based algorithms. The BM25 (Best Match 25) algorithm is an extension of TF-IDF [3, 7], and has become a cornerstone of the field of sparse search in NLP. BM25 has three primary components: Term Frequency (TF), Inverse Document Frequency (IDF), and Document Length Normalization. Unlike TF-IDF, where term frequency grows linearly, BM25 introduces a *saturation effect*. This means that the impact of term frequency on the score diminishes as the term appears more frequently in a document, and is calculated as

$$\mathrm{TF}_{\mathrm{BM25}}(t, d) = \frac{f(t, d) \cdot (k_1 + 1)}{f(t, d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)}$$

where $f(t, d)$ is the frequency of term t in document d. k_1 and b are tuning parameters (usually $k_1 \geq 1$ and $0 \leq b \leq 1$). $|d|$ is the length of document d. avgdl is the average document length in the corpus.

Similar to TF-IDF, BM25 uses IDF to weigh down the importance of terms that appear in many documents. However, BM25's IDF calculation is slightly different to prevent negative scores for very common terms, and is calculated as

$$\text{IDF}_{\text{BM25}}(t) = \log\left(\frac{df(t) + 0.5}{N - df(t) + 0.5} + 1\right)$$

where N is the total number of documents in the corpus, and $df(t)$ is the number of documents containing the term.

Combining the functions for TF and IDF as done in TF-IDF, the aggregate function for BM25 simply equates to the product of TF and IDF and accounting for document normalization, similar to TF-IDF, and is shown as such:

$$\text{BM25}(d, Q) = \sum_{q_i \in Q} \text{IDF}(q_i) \cdot \text{TF}_{\text{BM25}}(q_i, d)$$

where $f(q_i, d)$ is the frequency of term q_i in document d, $\text{IDF}(q_i)$ is the Inverse Document Frequency of term q_i, k_1 and b are tuning parameters (typically $k_1 \geq 1$ and $0 \leq b \leq 1$, $|d|$ is the length of document d, and avgdl is the average document length in the corpus.

4.4 SPLADE

SPLADE [11] is a neural retrieval model designed to improve the efficiency and effectiveness of sparse retrieval systems. SPLADE models the representation of text (either a document or a query) using a neural network, often based on transformers like BERT [14] and BERT's MLM (Masked Language Model). Similar to most previously discussed sparse algorithms, the primary goal of SPLADE, and its successor SPLADE v2 [12], is to convert text into sparse vector embeddings (SPLADE v3 has been configured and published in recent research [23], but is still in early evaluation phases and thus is not included in this study).

SPLADE and SPLADE v2 differ from core sparse search algorithms like TF-IDF and BM25, both of which are pure lexical traversal models, by utilizing transformers and dense vector embeddings. However, unlike dense retrieval algorithms and techniques, SPLADE leverages a sparsity-inducing mechanism that allows for selective activation of features that enables the conversion of dense embeddings to sparse vector embeddings, similar to how TF-IDF and BM25 focus on specific term frequencies.

The process begins with the input text (e.g., a document or query) being tokenized and passed through a transformer model. Each token in the text is represented as a high-dimensional vector, So if $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$ denotes the token embeddings for a sequence of n tokens, where $\mathbf{x}_i \in \mathbb{R}^d$ is the embedding of token i. The transformer applies self-attention mechanisms to generate contextualized embeddings. The self-attention mechanism computes attention scores and weighted sums of token embeddings. For a token i, the attention score with respect to token j is calculated as:

$$\text{score}(i, j) = \frac{(Q_i W_Q)(K_j W_K)^T}{\sqrt{d_k}}$$

where Q_i is the query vector, K_j is the key vector, W_Q and W_K are weight matrices, and d_k is the dimension of the key vectors. For the attention output, the weighted sum of value vectors V_j is computed as (*NOTE: softmax is an activation function as described in the next paragraph*):

$$\text{attention}(i) = \sum_j \text{softmax}\left(\text{score}(i, j)\right) \cdot V_j$$

After applying multiple layers of self-attention and feed-forward neural networks, the transformer generates dense embeddings $\mathbf{e}$ for the tokens in the sequence. Each embedding $\mathbf{e}_i$ is a vector in $\mathbb{R}^d$.

Next comes the process of converting the dense embeddings into sparse embeddings. SPLADE introduces a mechanism to perform this conversion, which involves applying sparsity constraints to the dense embeddings. The dense embeddings are passed through an activation layer to ensure non-negativity of entry values (frequency of a term is at minimum, 0). Prominent activation functions in Deep Learning include Sigmoid [24], Tanh [24, 25], Softmax [26], and ReLU (Rectified Linear Unit) [24–26]. A common choice is ReLU, calculated as $f(x) = \max(0, x)$, where x is an element of the dense embedding vector. To enforce sparsity, SPLADE employs additional regularization during training. The sparse representation $\mathbf{s}$ is obtained from the dense embedding $\mathbf{e}$ as follows:

$$\mathbf{s} = \text{ReLU}(W_s \cdot \mathbf{e})$$

where W_s is a learned transformation matrix. A sparsity-inducing penalty term is added to the loss function to encourage the model to produce sparse vectors. This is often done using L1 regularization:

$$L_{\text{sparsity}} = \lambda \sum_i |s_i|$$

where s_i are the components of the sparse vector $\mathbf{s}$, and λ is a hyperparameter controlling the strength of the sparsity constraint. The total loss function for SPLADE combines the retrieval loss with the sparsity penalty. Assuming a retrieval loss $L_{\text{retrieval}}$, the complete loss function is:

$$L = L_{\text{retrieval}} + \lambda \sum_i |s_i|$$

where $L_{\text{retrieval}}$ measures the performance of the retrieval task, ensuring that relevant documents are ranked higher for given queries. The term $\lambda \sum_i |s_i|$ penalizes the model based on the total magnitude of the sparse vector $\mathbf{s}$, encouraging sparsity.

5 Dense Retrieval

Dense retrieval, with the coupling of LLMs or any pretrained generative language models has proven to be a more than sound alternative to sparse retrieval. The core principle of dense retrieval is to promote contextual based searching, and does so by mapping text in a continuous vector space. Dense retrieval is composed of three key components: (1) embedding model, (2) indexing, and (3) vector similarity search.

5.1 Embedding

The foremost step in dense retrieval systems is to convert the documents and query into dense vector embeddings. This is most commonly done through the use of a neural network based transformer model, like BERT [14], RoBERTa [27], along with many other fine-tuned open source models on HuggingFace [28].

5.1.1 Tokenization

The embedding process is divided between query encoding and document encoding. The main corpus is typically tokenized (broken up into smaller units or partitions of words) using the tokenizer model associated with the transformer. For example, BERT uses WordPiece tokenization [14]. Standard transformer models compute such that each token is mapped to a unique numerical ID using a vocabulary base. Encoding methods vary, and can be performed a multitude of ways (cross-encoding, dual-encoding, etc.) [29].

5.1.2 Embedding Model Architecture

The main efficacy of dense retrieval systems is the language model that it contains. This language model, most commonly a generative model and one that is pre-trained on billions of parameters, are commonly used as the basis for embedding. The architecture involved a sequence of layers that process the input text to capture complex dependencies and contextual information across tokens, done in the encoding process [28].

5.1.3 Encoding Process

Post tokenization, the tokenized text is then fed into the transformer model's processing layers. These include the input layer, transformer layers, and the output layer.

This *input layer* then converts the tokens into dense vectors, using a learned embedding matrix. This matrix is essentially a lookup table where each row corresponds to a token and contains its vector representation in relation to the entire document. The stark contrast between dense and sparse retrieval stems from this step - the encoded vectors do not represent term frequency, but a specific weight in relation to the surrounding words and semantics. Since transformers do not inherently capture the order of tokens, positional encodings are added to the token embeddings. However, some research has shown that transformer models still capture context despite the absence of positional encodings [30]. These encodings provide information about the position of each token in the sequence, enabling the model to understand the order and structure of the input text, in addition to just context and meaning.

Once the tokens are embedded with their positional encodings, the sequence is passed through multiple *transformer layers*. Each transformer layer is composed of two main components: self-attention mechanisms and feed-forward neural networks. The self-attention mechanisms that allow the model to focus on different parts of the input sequence when processing each token. Self-attention computes a weighted sum of all token embeddings, where the weights are determined by the relevance of each token to the others. This helps the model capture contextual relationships and dependencies between tokens. After the attention mechanism performs its computations and calculates the weighted sum of all token embeddings, the output is passed through a feed-forward neural network (FNN) , which consists of linear transformations and non-linear activation functions (Softmax, Tanh, ReLU, etc.) [24–26]. Each transformer layer also includes *layer normalization* and *residual connections*. Layer normalization [31] stabilizes the training process, while residual connections [32] help preserve information and improve gradient flow through the network.

After the tokenized embeddings have been passed through a FNN, activation functions, and normalized, the embeddings then pass through the *output layer*. Often, for tasks like dense retrieval or text classification due to the use of a FNN, a *pooling operation* is often applied to obtain a single fixed-size vector representing the entire input text [33]. In BERT, special tokens like [CLS] and [SEP] play crucial roles in this process [34, 35]. The [CLS] token, placed at the beginning of the input, is designed to aggregate information from the entire sequence, and its final embedding is often used as the overall representation of the input text [35]. The [SEP] token serves to separate distinct segments of text, such as in question-answer pairs, enabling BERT to handle multiple sequences simultaneously [34]. Pooling methods like mean or max pooling across all token embeddings can be used [33], but the [CLS] token is a common and effective choice for summarizing the text in dense retrieval tasks.

5.2 Indexing

Indexing in dense retrieval is a crucial process that involves organizing and storing dense vector representations of documents or passages in a way that allows for efficient and effective retrieval. Given the high dimensionality of the vectors produced by embedding models, the indexing process is designed to facilitate rapid similarity searches (like nearest neighbor search) across potentially millions or billions of documents.

5.2.1 Vector Storage

Vector storage in indexing cannot be handled through traditional SQL and NoSQL databases. Due to their high dimensionality data traditional relational databases are not suitable for this purpose due to their inefficiency in handling complex vector operations. As a result, post-embedding, document vectors are stored in a *vector database* [36, 37]. This is a commonly used type of DB in dense indexing that supports rapid retrieval operations on high-dimensional data. The primary asset of vector DBs is the apparent efficiency min conducting similarity search between encoded query vectors and document embeddings that are contained within the vector DB. Note that the *index structure* also plays a crucial role in the efficacy of the index. The methodology by which the DB is quantized and segmented enables enhanced effi-

ciency when conducting similarity search and retrieving the relevant documents due to less computationally expensive parsing of vector embeddings.

5.3 Search - Vector Similarity

The final component in dense retrieval after embedding the text and creating a robust index, is the *search* step. The "search", or the actual retrieval part of dense retrieval systems is completed by computing *vector similarity* between the query vector and the document embeddings in the vector database. Vector similarity is a measure of how close or similar two vectors are to each other in a vector space. When any data is converted into embeddings (dense vector representations), the relationships and similarities between different pieces of data are encoded in the distances or angles between their corresponding vectors [38].

5.3.1 Distance Methods & Metrics

There are numerous ways to calculate the similarity between two vectors and may ways to measure or evaluate a distance metric. Common distance metrics for vector similarity include:

- Cosine Similarity [39–42]
- Euclidean Distance [41, 43]
- Dot Product [40]
- Manhattan Distance (L1 Norm) [41]
- Hamming Distance [44]
- Minkowski Distance [42]

Cosine Similarity is often preferred because it measures the angle between two vectors, effectively focusing on the direction rather than the magnitude. This is particularly useful when the scale of the vectors might differ, which is common in text embeddings or other high-dimensional data representations. The value of cosine similarity ranges from -1 to 1, where 1 indicates that two vectors are in the exact same direction (highly similar), 0 indicates orthogonality (no similarity), and -1 indicates opposite directions (dissimilar). This makes the metric intuitive to interpret. Due to its disregard for length and magnitude of the vector since they are normalized, Cosine Similarity places emphasis on semantic similarity, embodying the core of dense retrieval in RAG.

5.3.2 Approximate Nearest Neighbor (ANN)

The task of efficiently finding the most similar document vectors to a given query vector is crucial, especially when dealing with large-scale datasets. Given the high dimensionality often found in the embeddings of these datasets, performing an *exact* nearest neighbor search (k-nearest neighbor - KNN) can be computationally expensive and time-consuming [45, 46]. This is where Approximate Nearest Neighbor (ANN) algorithms come into play [46], providing a practical solution to this challenge by offering a balance between search accuracy and speed.

ANNs aim to quickly identify embedding vectors that are near to the query vector in the vector space (mapped by the vector DB and the index), albeit with a small sacrifice in precision, as compared to a KNN. Instead of exhaustively comparing the query vector to *every* vector in the dataset, ANNs accelerate the search process by *approximating* the nearest neighbors, thus enabling faster retrieval without a significant loss in relevance.

Several common ANN techniques are employed in dense retrieval systems, each with its unique approach to balancing accuracy and efficiency:

- Locality-Sensitive Hashing (LSH) [47]: LSH is a technique that hashes vectors into buckets such that similar vectors are more likely to fall into the same bucket. During a search, only vectors within the same or nearby buckets are considered, drastically reducing the number of comparisons needed [48] as compared to compare similar and dissimilar vectors in conjunction. LSH is particularly effective in high-dimensional spaces, where traditional methods struggle with the hurdle of high dimensionality.
- Hierarchical Navigable Small World (HNSW) Graphs [49]: HNSW builds a multi-layered graph where each node represents an embedding vector, and edges connect nodes that are likely to be near each other in the vector space. The search begins at a higher level of the graph and navigates downwards, progressively refining the search to find the nearest neighbors as it descends down each layer or level of the graph. This method offers both speed and accuracy, making it a popular choice for large-scale vector search tasks [49].
- Product Quantization (PQ) [50–52]: PQ compresses vectors into shorter codes that approximate their original distances. By reducing the dimensionality of the vectors and comparing these compressed representations, PQ enables faster searches while maintaining a reasonable level of accuracy. This method is particularly useful in scenarios where memory usage and computational resources are constrained [52].

The use of ANN in dense retrieval allows systems to handle millions or even billions of vectors efficiently, making it a key component in applications such as search engines, recommendation systems, and large-scale information retrieval tasks. While ANN sacrifices some accuracy compared to exact nearest neighbor search, the trade-off is often acceptable given the substantial gains in retrieval speed and scalability. This makes ANN an indispensable tool in the implementation of dense retrieval systems, enabling them to operate effectively in real-world scenarios where speed and scalability are paramount.

5.4 Cross Encoding vs Dual Encoding

The standard architecture and composition of modern dense retrieval systems fall under *Dual Encoding*, or *bi-encoding*, which is when the query and corpus are encoded separately into fix-sized vector representations using identical transformer models. After they are successfully encoded and the index is created, similarity search is conducted to compare the query vector to the vector embeddings stored in the DB. In *cross encoding* [53], the query and the documents are concatenated and processed together by a singular transformer model (e.g BERT [14]) and uses self-attention mechanisms to process this sequence. In each

layer of the transformer, self-attention computes attention scores between all pairs of tokens in the input sequence, regardless of whether the tokens belong to the query or the document, providing for a far more holistic perusal of the data and promotes greater semantic understanding. By processing the query and document together, cross-encoding can model complex relationships and dependencies between the two, which can prove to be increasingly improving in tasks where the relevance of a document depends on nuanced interactions between the query and various parts of the document or the information within the documents. However, the major compromise to the increased semantic understanding is that parsing through the entire dataset instead of simply identifying the approximately nearest documents is extremely computationally intensive. Since each query-document pair needs to be processed together, it requires running the transformer model once for every pair. This is becomes a bottleneck when dealing with large numbers of documents, and therefore prevents scalability for larger datasets, and has shown to be more suitable for reranking a smaller set of candidate documents. Although, recent research has emerged investigating various methods to accelerate the latency of cross encoder retrieval systems [54].

6 Hybrid Retrieval Architecture

Recent studies in retrieval systems and RAG document search processes have alluded to a novel methodology in RAG retrieval: Hybrid Retrieval. Hybrid retrieval, as shown by Figure 2 and BlendedRAG [3], is a process of coupling sparse and dense retrieval mechanisms and concatenating their respective results after holistic vector and output normalization. Figure 2 illustrates the primary differences between standard RAG architecture, shown in Figure 1, and RAG that incorporates hybrid retrieval techniques. Additionally, Figure 2 demonstrates the composition of the RAG model used for evaluation to illuminate the enhanced retrievals of this hybrid architecture. Furthermore, by focusing on the optimization of the sparse, dense, and encoder-decoder based retrieval models primary aim of highlighting the salient results of BlendedRAG [3].

From Figure 1 it's shown that standard RAG models follow a similar structure, beginning with querying and uploading documents for the creation of an index. Following the process of indexing (chunking and embedding), the retrieval system, which is generally split between either sparse of dense for allocation of computational resources and time complexity, conducts some form of similarity search or uses an ANN or Neural Network to compute the vectors located in the vector DB that are nearest to the query vector. After retrieval, the ordered documents get re-ranked repeatedly (either iteratively or recursively) and are input into the generation phase, in which the generative language model creates output as the final step. The main component in hybrid RAG that builds on standard RAG is the retriever and search functionality. While corpora are indexed the same way using a transformer model, hybrid RAG synchronizes sparse and dense retrieval techniques and concatenates their respective results.

Sparse retrieval methods, such as BM25 and TF-IDF, are effective at capturing exact matches and lexical overlap between the query and documents. On the other hand, dense retrieval methods, which rely on learned embeddings, excel at capturing semantic similarity even when the query and documents do not share the exact same vocabulary. By integrating and combining these approaches, hybrid retrieval aims to mitigate the weaknesses of each method when used in isolation, and emphasize the strengths of each method when used in conjunction, leading to a more robust retrieval system.

The concatenation of results from both retrieval strategies, as highlighted in Figure 2, involves a careful normalization process to ensure that the scores from sparse and dense retrievers are comparable. This is crucial because the raw scores generated by these methods can be on different scales and dimensions, potentially skewing the final results if not properly balanced when combined. Holistic vector normalization ensures that the results from both retrieval mechanisms contribute equitably to the final set of retrieved documents, preventing one method from dominating the other or from dimensionality inequalities from disrupting the reranking or generative processes that succeed retrieval.

Moreover, the hybrid RAG architecture optimizes not only the retrieval process but also the integration of these retrieved results into the generative phase. By feeding a more diverse and contextually rich set of documents into the encoder-decoder framework, the revised retrieval structure enhances the model's ability to generate more accurate and contextually relevant responses. This is particularly beneficial in scenarios where the query is ambiguous or requires information from multiple documents to generate a comprehensive answer. This improvement corresponds to the models increased ability to retrieve and integrate information that is both contextually relevant and semantically structured, addressing the limitations observed in models that rely solely on sparse or dense retrieval methods.

7 Experimental Method

To construct a base and efficacy-driven model to demonstrate the utility of hybrid retrieval in RAG, the components in accordance to Figure 2 are included (tokenizer, index, etc.). The process begins by loading SQuAD's dataset, initializing the transformer model, and constructing the QA generative pipeline. For the transformer model, HuggingFaces' top performing 7B model, MiniLMv6 is used [55], and for the QA pipeline, RoBERTa-base-SQuADv2 is used, which is a later version of RoBERTa [27]. Due to limitations of computational resources, the SQuAD dataset is narrowed down to a capable capacity. Then, two stages of evaluation take place: (1) evaluation the base model, (2) evaluating the hybrid model. During the evaluation of the base model, we do not conduct any hyperparameter tuning, and set the parameter α to a fixed value of 0.5. For the hybrid model evaluation, we conduct an iterative process of hyperparameter tuning, and conduct a ranking process of evaluation scores based on pairs of α and k values. The hybrid model returns the scores provided by the parameter pair that generate that greater accuracy.

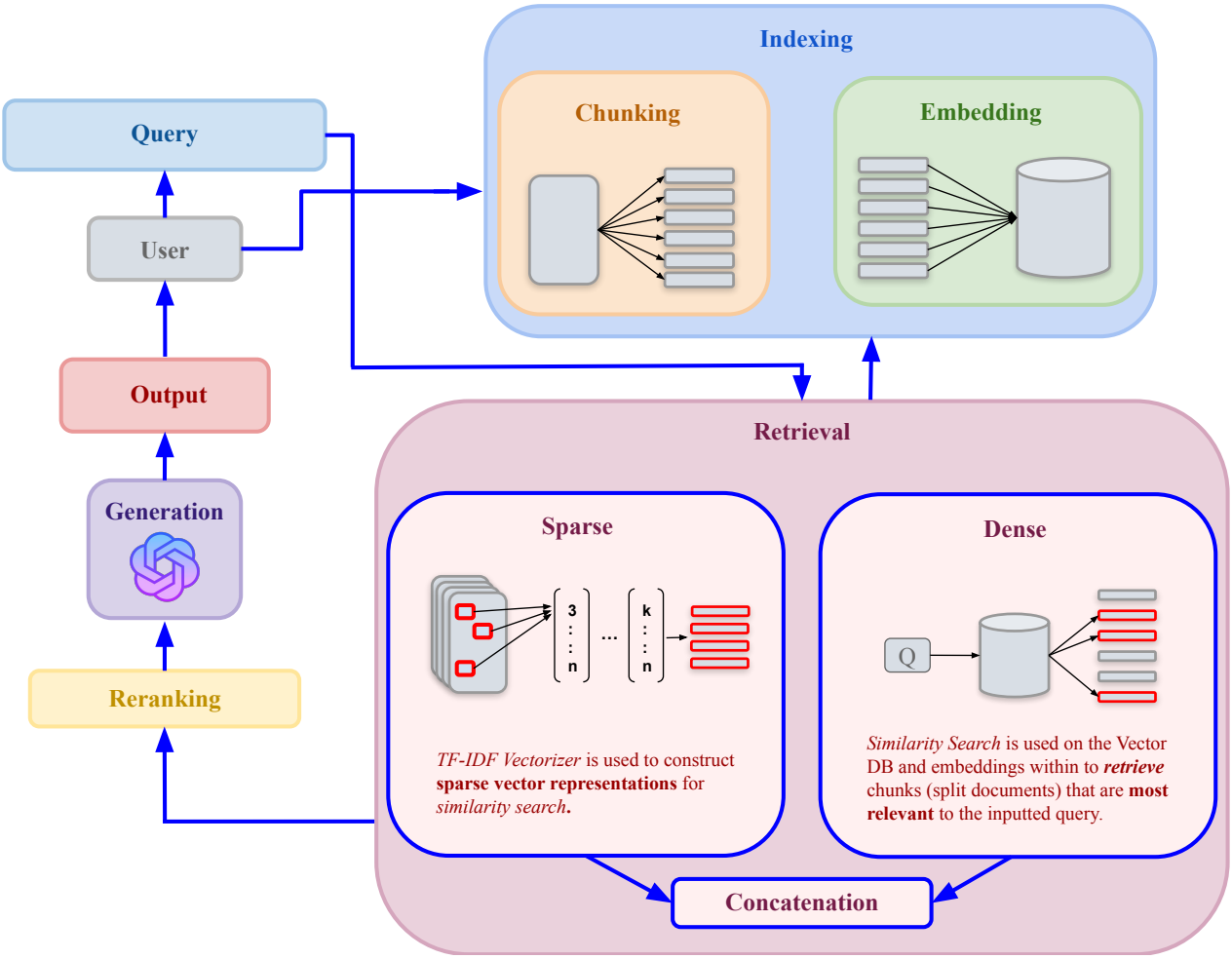

Figure 2: Representation of focused RAG Architecture with Hybrid Retrieval. Includes same processes as the standard RAG architecture shown in Figure 1 but clearly outlines a restructured, robust retrieval model. Example query and output are provided through the use of the constructured model using BERT and HuggingFace's MiniV6LM, simply to demonstrate the usage of a generative language model or LLM for language purposes.

8 Datasets & Evaluation

Evaluation is performed on two RAG architectures:

- Base: a standard RAG architecture with a QA pipeline and a transformer model to aid with semantic search.
- Hybrid: a hybrid model that builds on the base architecture by combining vectors returned from TF-IDF (sparse) with the vectors returned from the transformer model (dense).

To evaluate each of these models, the SQuAD dataset is used [8]. Furthermore, the models are each evaluated on SQuAD based on the listed evaluation metrics:

- Accuracy [56]: *accuracy* computes the fraction of questions for which the model's predicted answer matches the correct answer based on the provided dataset. This measures the model's holistic ability to produce correct responses across the dataset.
- F1 Score [57]: is the harmonic average between the values of *precision* and *recall* of a language model or classification model, described below.
- Precision / Recall [58, 58]: *precision* & *recall* both measure the ratio of true positives to true positives + false positives. In other words, the number of correct prediction to total positive predictions.
- ROUGE-L [59]: *ROUGE-L* measures the quality of the generated answers by examining the longest common subsequence (LCS) between the predicted and reference answers contained within the dataset. This metrics reflects the overall fluency and coherence of the model's answers in relation to the provided references.

These metrics have been chosen as the evaluation metric due to their simplicity in depicting the efficacy and performance of a RAG model with low computational costs and wide range use within the field of Retrieval Augmented Generation, and their low bias regarding its method for scoring baseline models that haven't been fine-tuned on Q&A datasets.

Evaluation Scores		
Metrics	**Base**	**Hybrid**
Accuracy	60%	64%
F1 Score	0.38	0.42
Precision	0.41	0.42
Recall	0.38	0.47
ROUGE-L	0.66	0.71

From the table above, the increase in model accuracy and F1 Score is 4% and 0.04, respectively, indicating improvement from the baseline in terms of performance on Generative Q&A. One point of important note is that neither models have been trained or fine-tuned on the SQuAD dataset. Furthermore, both precision and recall see improvements, indicating a marginal increase in correct predictions and accurate answers from the hybrid model as compared to the base model. Finally, the ROUGE score of the hybrid model substantially improves over the base model, indicating strong improvements in exact text and semantic retrieval (LCS), which as discussed, stems from improvements made due to the addition of sparse retrieval.

9 Conclusion

Hybrid retrieval architectures in RAG models have been proven to generate clear and improved results in RAG retrieval systems when coupled with larger scale LLMs and fine-tuned models, but not with smaller scale models. This paper demonstrates the improvements that can be made to smaller scale transformer and pretrained LM's using a hybrid approach. Prior to presenting the findings, the study provides a deep dive into retrieval techniques, mainly being sparse and dense retrieval, along with the most commonly used algorithms for each retrieval method, and examines each layer of those algorithms in detail and their relation to RAG architectures. The study then presents the constructed RAG models, both the baseline and the hybrid model, and elaborates on how the hybrid models utilizes the normalization and concatenation of the dense/sparse vectors. From the evaluation scores of the respective models, we provide conclusive evidence that hybrid retrieval approaches provides marginal improvements to smaller scale models, and that it remains a viable approach to supplement the current capabilities of LLMs.

10 Limitations

The primary goal of this study was to provide an overview on modern and widely-used retrieval architectures in RAG, and how combining retrieval architectures can enhance the efficacy and performance of a smaller scale language models. However, there were a few limitations and areas of future research, as listed and described below:

- Small Model Size: MiniLMv6, while one of HuggingFace's top performing sentence transformer models, it's semantic capabilities and literary output are incomparable to LLMs like Llama-70B or GPT-4o, which contain billions of parameters.
- Range of Testing: for this study, only the SQuAD Q&A dataset was used. Other datasets like Google's NQ dataset or RAGAS could be used for wider evaluation of the constructed models. Furthermore, more parameters and evaluation metrics could be used to provide deeper and more detailed depictions of model improvements and/or degradation.
- Fine-tuning: fine-tuning both models on the datasets used for evaluation or on general QA fine-tuning datasets would enhance baseline model performance and would not only accentuate the changes due to hybrid retrieval, but would increasing performance for that retrieval system as well.
- Challenge of Resources: the primary challenge whilst conducting this study and engineering the baseline and hybrid models was the lack of computational resources to use more advanced technologies and more capable models.

References

[1] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, Q. Guo, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," 2024.

[2] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2021.

[3] K. Sawarkar, A. Mangal, and S. R. Solanki, "Blended rag: Improving rag (retriever-augmented generation) accuracy with semantic search and hybrid query-based retrievers," 2024.

[4] F. Cuconasu, G. Trappolini, F. Siciliano, S. Filice, C. Campagnano, Y. Maarek, N. Tonellotto, and F. Silvestri, "The power of noise: Redefining retrieval for rag systems," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR 2024, ACM, July 2024.

[5] B. Reichman and L. Heck, "Retrieval-augmented generation: Is dense passage retrieval retrieving?," 2024.

[6] X. Wang, Z. Wang, X. Gao, F. Zhang, Y. Wu, Z. Xu, T. Shi, Z. Wang, S. Li, Q. Qian, R. Yin, C. Lv, X. Zheng, and X. Huang, "Searching for best practices in retrieval-augmented generation," 2024.

[7] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[8] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," 2016.

[9] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, M. Kelcey, J. Devlin, K. Lee, K. N. Toutanova, L. Jones, M.-W. Chang, A. Dai, J. Uszkoreit, Q. Le, and S. Petrov, "Natural questions: a benchmark for question answering research," *Transactions of the Association of Computational Linguistics*, 2019.

[10] S. E. Robertson, S. Walker, M. Hancock-Beaulieu, and M. Gatford, "Okapi at trec-3," in *Proceedings of the Third Text REtrieval Conference (TREC-3)*, (Gaithersburg, MD), pp. 109–126, National Institute of Standards and Technology (NIST), 1995.

[11] T. Formal, B. Piwowarski, and S. Clinchant, "Splade: Sparse lexical and expansion model for first stage ranking," 2021.

[12] T. Formal, C. Lassance, B. Piwowarski, and S. Clinchant, "Splade v2: Sparse lexical and expansion model for information retrieval," 2021.

[13] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych, "Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models," 2021.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[15] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," 2019.

[16] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. tau Yih, "Dense passage retrieval for open-domain question answering," 2020.

[17] Z. Dong, J. Ni, D. M. Bikel, E. Alfonseca, Y. Wang, C. Qu, and I. Zitouni, "Exploring dual encoder architectures for question answering," 2022.

[18] W. Jiang, S. Zhang, B. Han, J. Wang, B. Wang, and T. Kraska, "Piperag: Fast retrieval-augmented generation via algorithm-system co-design," 2024.

[19] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. van den Driessche, J.-B. Lespiau, B. Damoc, A. Clark, D. de Las Casas, A. Guy, J. Menick, R. Ring, T. Hennigan, S. Huang, L. Maggiore, C. Jones, A. Cassirer, A. Brock, M. Paganini, G. Irving, O. Vinyals, S. Osindero, K. Simonyan, J. W. Rae, E. Elsen, and L. Sifre, "Improving language models by retrieving from trillions of tokens," 2022.

[20] Z. Liao, H. Yu, J. Li, J. Wang, and W. Zhang, "D2llm: Decomposed and distilled large language models for semantic search," 2024.

[21] R. Nogueira and K. Cho, "Passage re-ranking with bert," 2020.

[22] A. Mao, M. Mohri, and Y. Zhong, "Cross-entropy loss functions: Theoretical analysis and applications," 2023.

[23] C. Lassance, H. Déjean, T. Formal, and S. Clinchant, "Splade-v3: New baselines for splade," 2024.

[24] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," 2022.

[25] M. M. Hasan, M. A. Hossain, A. Y. Srizon, and A. Sayeed, "Talu: A hybrid activation function combining tanh and rectified linear unit to enhance neural networks," 2023.

[26] K. Shen, J. Guo, X. Tan, S. Tang, R. Wang, and J. Bian, "A study on relu and softmax in transformer," 2023.

[27] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019.

[28] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Huggingface's transformers: State-of-the-art natural language processing," 2020.

[29] D. Rau, S. Wang, H. Déjean, and S. Clinchant, "Context embeddings for efficient answer generation in rag," 2024.

[30] A. Haviv, O. Ram, O. Press, P. Izsak, and O. Levy, "Transformer language models without positional encodings still learn positional information," 2022.

[31] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016.

[32] S. Xie, H. Zhang, J. Guo, X. Tan, J. Bian, H. H. Awadalla, A. Menezes, T. Qin, and R. Yan, "Residual: Transformer with dual residual connections," 2023.

[33] H. Gholamalinezhad and H. Khosravi, "Pooling methods in deep neural networks, a review," 2020.

[34] X. Chen, Y. Wang, Y. Du, S. Hassoun, and L.-P. Liu, "On separate normalization in self-supervised transformers," 2023.

[35] L. Wu, W. Zhang, T. Jiang, W. Yang, X. Jin, and W. Zeng, "[cls] token is all you need for zero-shot semantic segmentation," 2023.

[36] Y. Han, C. Liu, and P. Wang, "A comprehensive survey on vector database: Storage and retrieval technique, challenge," 2023.

[37] J. J. Pan, J. Wang, and G. Li, "Survey of vector database management systems," 2023.

[38] S. Bruch, *Foundations of Vector Retrieval*. Springer Nature Switzerland, 2024.

[39] H. Steck, C. Ekanadham, and N. Kallus, "Is cosine-similarity of embeddings really about similarity?," in *Companion Proceedings of the ACM on Web Conference 2024*, WWW '24, ACM, May 2024.

[40] C. Luo, J. Zhan, L. Wang, and Q. Yang, "Cosine normalization: Using cosine similarity instead of dot product in neural networks," 2017.

[41] F. Tessari and N. Hogan, "Surpassing cosine similarity for multidimensional comparisons: Dimension insensitive euclidean metric (diem)," 2024.

[42] E. Garcia-Morato, M. J. Algar, C. Alfaro, F. Ortega, J. Gomez, and J. M. Moguerza, "A general framework for distributed approximate similarity search with arbitrary distances," 2024.

[43] M. Hoffmann and F. Noé, "Generating valid euclidean distance matrices," 2019.

[44] B. Nikolić and B. Šobot, "Measures of string similarities based on the hamming distance," 2022.

[45] P. Cunningham and S. J. Delany, "k-nearest neighbour classifiers - a tutorial," *ACM Computing Surveys*, vol. 54, p. 1–25, July 2021.

[46] R. Chen, B. Liu, H. Zhu, Y. Wang, Q. Li, B. Ma, Q. Hua, J. Jiang, Y. Xu, H. Deng, and B. Zheng, "Approximate nearest neighbor search under neural similarity metric for large-scale recommendation," 2022.

[47] O. Jafari, P. Maurya, P. Nagarkar, K. M. Islam, and C. Crushev, "A survey on locality sensitive hashing algorithms and their applications," 2021.

[48] S. Miao, Z. Lu, M. Liu, J. Duarte, and P. Li, "Locality-sensitive hashing-based efficient point transformer with applications in high-energy physics," 2024.

[49] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," 2018.

[50] Q. Yue, X. Xu, Y. Wang, Y. Tao, and X. Luo, "Routing-guided learned product quantization for graph-based approximate nearest neighbor search," 2023.

[51] A. F. AbouElhamayed, A. Cui, J. Fernandez-Marques, N. D. Lane, and M. S. Abdelfattah, "Pqa: Exploring the potential of product quantization in dnn hardware acceleration," 2024.

[52] Z. Li, J. Ji, Y. Ge, W. Hua, and Y. Zhang, "Pap-rec: Personalized automatic prompt for recommendation language model," 2024.

[53] H. Déjean, S. Clinchant, and T. Formal, "A thorough comparison of cross-encoders and llms for reranking splade," 2024.

[54] A. V. Petrov, S. MacAvaney, and C. Macdonald, *Shallow Cross-Encoders for Low-Latency Retrieval*, p. 151–166. Springer Nature Switzerland, 2024.

[55] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," 2020.

[56] P. Chen, J. Ye, G. Chen, J. Zhao, and P.-A. Heng, "Robustness of accuracy metric and its inspirations in learning with noisy labels," 2020.

[57] M. Sitarz, "Extending f1 metric, probabilistic approach," *Advances in Artificial Intelligence and Machine Learning*, vol. 03, no. 02, p. 1025–1038, 2023.

[58] F. L. Bronnec, A. Verine, B. Negrevergne, Y. Chevaleyre, and A. Allauzen, "Exploring precision and recall to assess the quality and diversity of llms," 2024.

[59] S. Takeshita, S. P. Ponzetto, and K. Eckert, "Rouge-k: Do your summaries have keywords?," 2024.