UNIVERSIDADE FEDERAL DE GOIÁS (UFG)

INSTITUTO DE INFORMÁTICA (INF)

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO (PPGCC)

ALEX MICHAEL CHIHURURU

Proposal and Evaluation of Efficient Pruning Approaches for Multi-Vector Representation in Passage Retrieval

GOIÂNIA

2025

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA) PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES

E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo das Teses e Dissertações disponibilizado na BDTD/UFG é de responsabilidade exclusiva do autor. Ao encaminhar o produto final, o autor(a) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do material bibliográfico

[X] Dissertação [ ] Tese [ ] Outro*:_____

*No caso de mestrado/doutorado profissional, indique o formato do Trabalho de Conclusão de Curso, permitido no documento de área, correspondente ao programa de pós-graduação, orientado pela legislação vigente da CAPES.

Exemplos: Estudo de caso ou Revisão sistemática ou outros formatos.

2. Nome completo do autor

Alex Michael Chihururu

3. Título do trabalho

Proposal and Evaluation of Efficient Pruning Approaches for Multi-Vector Representation in Passage Retrieval

4. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador)

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante:
a) consulta ao(à) autor(a) e ao(à) orientador(a);
b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo da tese ou dissertação.
O documento não será disponibilizado durante o período de embargo.
Casos de embargo:
- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

Obs. Este termo deverá ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Thierson Couto Rosa**, **Professor do Magistério Superior**, em 26/06/2025, às 22:12, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Alex Michael Chihururu**, **Discente**, em 27/06/2025, às 05:33, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5464949** e o código CRC **040A17A0**.

Referência: Processo nº 23070.022902/2025-35 SEI nº 5464949

ALEX MICHAEL CHIHURURU

Proposal and Evaluation of Efficient Pruning Approaches for Multi-Vector Representation in Passage Retrieval

Dissertation submitted to the Programa de Pós-Graduação em Ciência da Computação of the Instituto de Informática of the Universidade Federal de Goiás, as a partial requirement for obtaining the title of Master in Ciência da Computação.
Concentration area: Ciência da Computação
Research Line: Sistemas Inteligentes e Aplicações

Mentor: Prof. Dr. Thierson Couto Rosa

GOIÂNIA

2025

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Chihururu, Alex Michael
Proposal and Evaluation of Efficient Pruning Approaches for Multi Vector Representation in Passage Retrieval [manuscrito] / Alex Michael Chihururu. - 2025.
cxxv, 125 f.

Orientador: Prof. Dr. Thierson Couto Rosa.
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2025.
Bibliografia.
Inclui gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. Pruning methods. 2. Multivector retrieval model. 3. Bi-encoder. 4. Information retrieval. I. Rosa, Thierson Couto, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

ATA DE DEFESA DE DISSERTAÇÃO

Ata nº **16** da sessão de Defesa de Dissertação de **Alex Michael Chihururu**, que confere o título de Mestre em Ciência da Computação, na área de concentração em Ciência da Computação.

Aos treze dias do mês de junho de dois mil e vinte e cinco, a partir das catorze horas, via sistema de webconferência da RNP, realizou-se a sessão pública de Defesa de Dissertação intitulada **"Proposal and Evaluation of Efficient Pruning Approaches for Multi-Vector Representation in Passage Retrieval"**. Os trabalhos foram instalados pelo Orientador, Professor Doutor Thierson Couto Rosa (INF/UFG) com a participação dos demais membros da Banca Examinadora: Professor Doutor Wladmir Cardoso Brandão (PUC/MINAS), membro titular externo; Professor Doutor Wellington Santos Martins (INF/UFG), membro titular interno. A realização da banca ocorreu por meio de videoconferência. Durante a arguição os membros da banca não fizeram sugestão de alteração do título do trabalho. A Banca Examinadora reuniu-se em sessão secreta a fim de concluir o julgamento da Dissertação, tendo sido o candidato **aprovado** pelos seus membros. Proclamados os resultados pelo Professor Doutor Thierson Couto Rosa, Presidente da Banca Examinadora, foram encerrados os trabalhos e, para constar, lavrou-se a presente ata que é assinada pelos Membros da Banca Examinadora, aos treze dias do mês de junho de dois mil e vinte e cinco.

TÍTULO SUGERIDO PELA BANCA

Documento assinado eletronicamente por **Wladmir Cardoso Brandão**, **Usuário Externo**, em 13/06/2025, às 16:00, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Wellington Santos Martins**, **Professor do Magistério Superior**, em 13/06/2025, às 16:00, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Thierson Couto Rosa**, **Professor do Magistério Superior**, em 13/06/2025, às 16:00, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Alex Michael Chihururu**, **Usuário Externo**, em 13/06/2025, às 16:08, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5434462** e o código CRC **4DC4273A**.

Alex Michael Chihururu

Graduated in Computer Engineering from Universidade Zambeze and in Human Resource Management from Universidade Licungo, holding certifications in Scientific Communication, 5G Technology, Scientific Methodology with AI, and Python Programming. During a Master's in Computer Science at UFG with a CAPES scholarship through GCUB, served as a teaching assistant in Formal Languages in Computer Engineering and is a University Assistant at Universidade Púnguè in Mozambique. Currently developing solutions for ad hoc information retrieval problems.

I dedicate this work to Catarina Michael Chihururu [in memory], Nelson Michael Chihururu [in memory].

Acknowledgements

I would like to express my deep gratitude to all those who contributed to making this achievement a reality. First and foremost, to my parents, Michael Chihururu and Isabel Francisco Saene, for the life they gave me, their tireless effort, and their unwavering support at all times. If today I am who I am and have reached what I have, I owe it to you. Thank you so much for everything.

To my family and friends, who have stood by my side at every stage, offering strength, encouragement, and celebrating victories with me. You have been my safe haven during challenging moments and my greatest motivation to keep moving forward.

To my colleagues, who welcomed me with so much warmth and friendship since my arrival in Brazil. Adapting to a new environment was significantly smoother thanks to the support of each of you. Especially to those who came from Mozambique, for creating a support network that made all the difference.

To my mentor, Thierson Couto Rosa, for his dedication, patience, and constant guidance throughout the entire process of developing this dissertation. Your support was crucial not only for my academic growth but also for my personal growth. I am immensely grateful for your availability and commitment.

To the professors, who shared their knowledge and skills, directly contributing to the development of this work. This dissertation is a reflection of the teaching and dedication of each of you.

Finally, to all those who, in some way, directly or indirectly, contributed to this journey, my sincere thanks. This achievement would not have been possible without the support and collaboration of each one of you. Thank you so much!

Abstract

Chihururu, Alex Michael. **Proposal and Evaluation of Efficient Pruning Approaches for Multi-Vector Representation in Passage Retrieval**. Goiânia, 2025. 125p. MSc. Dissertation. Programa de Pós-graduação em Ciência da Computação, Instituto de Informática, Universidade Federal de Goiás.

Multi-vector retrieval models employ bi-encoders to generate contextualized embeddings for queries and passages, and have proven highly effective in capturing fine-grained token-level interactions. Models such as ColBERT, ColBERTv2, and PLAID leverage all token-level output vectors from the encoder to accurately model query-passage relationships. However, storing dense vectors for every token in each passage results in substantial memory overhead. Additionally, query latency is significantly affected by the computational cost of computing inner products between each query token and all passage tokens to obtain similarity scores. In this work, we explore pruning techniques applied to passage vectors produced by PLAID, aiming to remove less important token vectors to improve memory efficiency and reduce query processing time, with minimal impact on retrieval effectiveness. We propose two novel pruning methods: MLM Max with Token Reordering (MMTR) and TF-IDF pruning. We conducted extensive experiments on both in-domain and zero-shot (out-of-domain) datasets, following best-practice evaluation protocols. Our results show that MMTR consistently yields the smallest effectiveness drop compared to the original, unpruned PLAID model. We observe that retaining 50% of the passage token embeddings provides the best trade-off between effectiveness, index size, and latency across most datasets. Interestingly, on certain out-of-domain datasets, pruning acts as a form of noise reduction—where retaining only 25% of the token embeddings leads to improved retrieval performance over the full, unpruned index.

Keywords

Pruning methods, Multi-vector retrieval models, Bi-encoders, Information retrieval.

Resumo

Chihururu, Alex Michael. **Representação multivetorial efetivo e eficiente para recuperação de passagens.**. Goiânia, 2025. 125p. Dissertação de Mestrado. Programa de Pós-graduação em Ciência da Computação, Instituto de Informática, Universidade Federal de Goiás.

Modelos de recuperação multivetorial empregam codificadores duplos para gerar embeddings contextuais para consultas e passagens e têm se mostrado altamente efetivos na captura de interações ricas em nível de token. Modelos como ColBERT, ColBERTv2 e PLAID aproveitam todos os vetores de saída em nível de token do codificador para modelar com precisão as relações consulta-passagem. No entanto, armazenar vetores para cada token de passagem impõe requisitos significativos de memória de armazenamento. Além disso, a latência da consulta é fortemente impactada pelo custo computacional da execução de operações de produto interno entre cada embedding de token de consulta e todos os embeddings de token de passagem para calcular pontuações de similaridade agregadas. Neste trabalho, investiga-se técnicas de poda aplicadas a embeddings de passagens produzidos por PLAID, com o objetivo de remover embeddings de tokens menos importantes, a fim de aumentar a eficiência da memória e reduzir o tempo de processamento de consultas, ao custo de uma possível pequena queda na efetividade. Propo-se dois novos métodos de poda: MLM Max com Reordenação de Tokens (MMTR) e TF-IDF pruning. Realiza-se experimentos extensivos em conjuntos de dados dentro e fora do domínio, seguindo os protocolos de avaliação recomendados. Os resultados mostram que o MMTR atinge consistentemente a menor queda na efetividade em comparação com o modelo PLAID sem poda. Além disso, a retenção de 50% dos embeddings de tokens de passagem oferece a melhor solução de compromisso entre efetividade de recuperação, tamanho do índice e latência da consulta na maioria dos conjuntos de dados. No entanto, para certos conjuntos de dados fora do domínio, a poda também atua como uma forma de redução de ruído. Nesses casos, mesmo quando apenas 25% dos vetores do PLAID são retidos, o modelo podado supera o PLAID sem poda, em efetividade de recuperação.

Palavras—chave

Métodos de poda, Modelos de recuperação multi-vetoriais, Bi-codificadores, Recuperação de informação.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

1.1 Introduction

Traditional passage information retrieval predominantly rely on an exact matching model[1] such as BM25 [70], using an efficient inverted index for retrieving passages in response to user queries. Although very efficient, exact matching models suffer from "vocabulary mismatch" problem [19, 39, 3], which arises when users formulate queries using terms or expressions that differ from those present in the relevant passages. These models are also known as *sparse retrievers* because they model queries and passages as vectors in the vocabulary space which are sparse given that only a fraction of the vocabulary occurs in both queries and documents.

Recent advancements in dense vector representations have enabled the modeling of terms within specific textual contexts through Pretrained Language Models (PLMs), such as Transformers [80]. Transformer-based ranking models have demonstrated significant performance improvements by mapping queries and passages into contextualized representations. These models reduce the vocabulary mismatch problem by performing soft matching in the vector space. They also leverage attention mechanisms to focus on critical parts of both queries and passages, enabling more accurate relevance estimation.

One of the uses of PLMs in Information Retrieval is as *dense retrievers*, also known as *bi-encoders* [29, 27, 44, 30]. Bi-encoders generate dense vector representations (embeddings) for passage tokens during index construction. At query time, the text composing the query is fed to the PLM and the query token embeddings are compared to document token embeddings stored in the index by means of a similarity measure (usually the cosine measure). The passages are ranked in decreasing order of their similarity with the query.

Models based on the bi-encoder architecture can be categorized into two types: single-vector models and multi-vector models. Single-vector models [29] use only one

[1] A ranking model in which terms from passages and terms from queries had to match exactly to contribute to a relevance score.

vector per query and passage to perform the retrieval, usually the vector corresponding to the [CLS] token[2]. In contrast, multi-vector retrieval models [30, 19, 74, 73] utilize multiple token vectors per query and passage.

1.1.1 Motivation

Multi-vector retrieval models, particularly those derived from ColBERT [30], leverage all encoder output tokens to capture fine-grained token-level interactions between queries and passages. While these models have demonstrated state-of-the-art effectiveness, they introduce two major computational bottlenecks. First, storing an embedding for every passage token leads to large and often impractical indexes [1, 3]. Second, relevance scoring requires computing similarities between multiple query and passage token embeddings, which considerably impacts query response times. Although ColBERTv2 [74] and PLAID [73] have much enhanced both time and space efficiency, these challenges still hinder the scalability and practical deployment of dense multi-vector models in end-to-end retrieval systems.

On the other hand, some authors have concluded that there are differences in importance among token vectors of a passage [1, 32, 41]. As consequence, some static methods have been investigated to prune less important token vectors to reduce both index size and query latency. For instance, Lassance et al. [32] compared different criteria for static pruning passage token vectors produced by ColBERT on the MS-MARCO dataset [55] and were able to prune 30% of ColBERT's index, incurring marginal effectiveness loss. However, the authors did not study the query latency impact when some vectors of passage or query are pruned in end-to-end retrieval systems. Liu et al. [41] compared distinct static pruning techniques on the MS-MARCO and BEIR datasets [77] with ColBERTv2 and COIL [19], and achieved a marginal loss in effectiveness when retaining 75% of the vectors. They also explored pruning query vectors in the retrieval stage and concluded that query latency can be reduced by a factor of 1.3 with little loss of effectiveness.

To the best of our knowledge, no prior work has analyzed the impact of pruning PLAID (Performance-optimized Late Interaction Driver) [73] output vectors on effectiveness, index size, and especially query latency. PLAID introduced a technique that avoids exhaustively scoring all passages retrieved via nearest-neighbor search by using centroids to identify high-scoring passages and eliminate weaker candidates. As a result, PLAID

[2]The [CLS] token is the first token in every input sequence processed by BERT. Its contextual embedding serves as an aggregate representation of the entire sequence. In ranking tasks—especially in single-vector models—the output vector corresponding to the [CLS] token is used to represent both the document and the query at a global level. The similarity between the query and the document is then computed using the inner product of their respective [CLS] vectors produced by BERT.

was able to reduce ColBERTv2 query latency by a factor of seven on a GPU and 45 times on a CPU. Thus, how pruning methods can enhance memory and especially the query latency of PLAID is still an open question.

1.1.2 Our Proposal

In this paper, we propose two novel approaches to prune less important passage token embeddings and apply them to PLAID vectors. The first approach uses the TF-IDF method, which removes passage token vectors corresponding to tokens with low TF-IDF values. The second method, called MLM Max with Token Reordering (MMTR), involves three steps: (i) First of all, each passage vector is passed through an MLM head, producing a distribution over BERT's vocabulary, as described by Ram et al. [67]; (ii) Second, all the projections are aggregated using max pooling to generate token scores for the passage; (iii) a token reordering strategy is used to avoid indexing multiple vectors of the same token, thereby improving token diversity. We compared the proposed methods against three pruning techniques previously investigated in the literature: First [32, 41], IDF [1, 32, 41], and Attention-based pruning [41]. These pruning methods are described in Section 2.8 and 4.4.

This study aimed to evaluate all five pruning strategies in terms of their ability to improve the efficiency of PLAID while not degrading its retrieval effectiveness. In this context, efficiency refers to both the reduction in the memory footprint of the pruned index and the decrease in query latency. Additionally, we aimed to analyze the sensitivity of each pruning method to varying the ratios of vectors maintained in the index and to assess their performance on both in-domain[3] and out-of-domain datasets.[4] Therefore, an empirical study was undertaken to address the subsequent research questions:

RQ1: What differences exist in the effectiveness of various pruning methods when applied to PLAID output vectors?

RQ2: Do different pruning methods vary in how they affect the trade-off between effectiveness and efficiency when applied to PLAID output vectors?

RQ3: Is there an approximate optimal ratio α for retaining PLAID output vectors that achieves a good balance between effectiveness and efficiency in both in-domain and out-of-domain datasets?

To evaluate our and previous methods in light of our research questions, we conducted experiments on two in-domain datasets: MS-MARCO and TREC DL 2019,

[3] In-domain data correspond to those which part o of the data were used for training PLAID, maintaining a similar distribution and characteristics.

[4] Out-of-domain data introduce unseen contexts, differing from those present in the training data, also known as zero-shot data and serve to evaluate the model's generalization ability.

and on the zero-shot BEIR dataset collection. Similar to Liu et al. [41], we experimented with three remaining ratios $\alpha \in \{25\%, 50\%, 75\%\}$, that determine the percentage of token vectors to be retained in each passage. We follow appropriate experimental protocols suggested in the literature [11, 21, 79, 66] for comparing multiple methods over a unique dataset (in the case of in-domain datasets) and over multiple datasets (in the case of out-of-domain datasets).

Our experiments show that there are statistical significant differences among the pruning methods in terms of effectiveness. For instance, MMTR was the most stable method, being more effective or tying with the best methods in most of the scenarios analyzed (considering different datasets and retaining ratios). Also, the methods First and Attention were shown to be less effective than MMTR in all datasets for the remaining ratios of 50% and 75%.

Regarding the trade-off between effectiveness and efficiency, we found that the remaining ratios affect differently in-domain and outo-of-domain datasets. In the case of in-domain datasets the 50% ratio corresponds to the best balance trade-off. For instance, at this ratio, MMTR was able to reduce 1.7x both the index size and query latency on the MS-MARCO dataset with a drop of only 2.26% in MRR@10 and of 5.1% in R@100 (Recall@100). In the case of the out-of-domain datasets, although the remaining ratio 50% is the best for the majority of the datasets, it is possible to obtain the best balance by maintaining only 25% of the PLAID's vector in the index.

1.1.3 Main Contributions

In sum, the main contribution of this work is as follows:

- The introduction of two static pruning methods (TF-IDF and MMTR) to be applied to the set of passage token vectors output by PLAID [73].
- An experimental comparison of the proposed methods with baseline (Attention, First and IDF), following recommended experimental protocols that allowed to show important differences in effectiveness among pruning methods not reported previously.
- A detailed analysis of the trade-off between effectiveness and efficiency of the pruning methods investigated when comparing to the non-pruning of PLAID vectors.
- An ablation analysis of the MMTR method.

1.1.4 Dissertation Organization

The remainder of this dissertation is organized as follows: Chapter 2 presents background and related work, highlighting main breakthroughs in the information retrieval area and most recent works in dense retrievers, it also explains some important

concepts around the methods used in this dissertation. Chapter 3 introduces the two new methods for pruning multi-vectors. Chapter 4 presents the datasets, the evaluation metrics, the baselines and the experimental protocol used in our experiments. Chapter 5 presents the empirical results and discus the main findings of this work and significant implications. Chapter 6 presents our conclusion and suggestions for future work.

CHAPTER 2

Background and Related Work

In this chapter we present the basic concepts necessary for understanding our dissertation. According to [22], IR models are classified into traditional models, discussed in Section 2.1, deep learning models, discussed in Section 2.2, and Transformer-based models, presented in Section 2.3. The literature review by Fan et al. [13] focuses on pre-trained language models applied to IR. The authors categorized existing IR models in this context into dense models (presented in Section 2.4), sparse models (in Section 2.5), and hybrid models (in Section 2.6). Finally, we review different approaches for enhancing the late interaction method proposed in ColBERT [30], including a review of static pruning techniques [32, 1, 41] that have been applied to reduce both the number of document token vectors stored and query processing time.

2.1 Traditional Ad Hoc Information Retrieval

Information Retrieval (IR) refers to the process of obtaining relevant information from a large collection of data, in response to a user query. Traditional IR models, also known as "keyword search", aim to match queries with documents, primarily through exact term matching, where the frequency of query terms in the document is a predominant factor in the document's relevance score [39]. These models can be broadly classified into four categories: boolean models, vector space models, language models, and probabilistic models [81]. Central to these approaches is the *bag-of-words (BoW)* assumption, where texts—whether queries or documents—are treated as unordered sets of words by not taking grammar, syntax, and word order into consideration [22].

2.1.1 Boolean Model

The boolean model is an IR model based on set theory and boolean algebra. Queries correspond to Boolean logic expressions, consisting of words separated by logical operators such as AND, OR, and NOT. The goal of the model is to find documents that satisfy the presence (or absence) of terms specified by the logical operations between the

query terms. Thus, the user can require that retrieved documents contain all query terms by using the AND operator between terms, the occurrence of at least one of several query terms (using the OR operator), or not occurrence of some query terms by using the NOT operator.

Boolean models are efficient, easy to implement, and perform well in certain situations. They have been used in IR systems for a long time and continue to be commonly employed in advanced search options available in various search engines. However, these models only retrieve documents that contain the combination of terms specified in the boolean expression. They are not truly ranking models, as they do not assign a relevance value to documents. Therefore, the boolean model does not allow the generation of a ranked list for the user but only a list of documents that satisfy the conditions specified in the query.

Extended Boolean Model

The extended boolean model aims to overcome the limitations of the basic Boolean model by incorporating term matching and weighting functions, introducing the notion of relative importance to term occurrences in specific parts of the document. In some document collections, there is a standard structure described by metadata, which typically includes fields such as publication date, author, title, abstract, and body text. The zoning technique allows associating different weights to document metadata within the boolean model. For example, consider a collection where documents have the following zones (similar to fields) with their respective weights: author with a weight of 0.2, title with a weight of 0.3, and body with a weight of 0.5. Suppose the term "UFG" appears only in the title and body of a document d, and the presence of the term is signed to 1 and absence to 0. The weight of d relative to this term is calculated as:

$$p(d, \text{UFG}) = 0.2 \times 0 + 0.3 \times 1 + 0.5 \times 1 = 0.8 \tag{2-1}$$

Thus, the extended boolean model can capture the concept of document ranking by weighting the relevance of terms according to their occurrences in different zones of the document. However, the notion of relative importance depends on the collection's structure, which may limit its application in collections with different structures.

2.1.2 Weighting Schemes

Since both queries and documents are composed of terms, proposals have emerged to use statistical information derived from term frequency and distribution to compute a relevance score for each document with respect to (w.r.t) a given query. The

method for calculating term weights depends on the model used, but most methods take into account the following factors [39, 13]:

- Term Frequency (TF): the number of occurrences of a term in a document.
- Inverse Document Frequency (IDF): the inverse of the number of documents in the collection that contain the term—it measures the rarity of the term in the collection. The rarer the term (higher IDF), the more discriminative it is in the documents where it occurs.
- Document Length: the size of the document in which the term occurs.

The most commonly used method in IR for calculating term weights in a document is the multiplication of TF by IDF, allowing a score to be calculated for a document d w.r.t a query q. The score is calculated as follows:

$$\text{Score} = \sum_{t \in q} TF(t,d) \times IDF(t) \tag{2-2}$$

where q is the query, $TF(t,d)$ is the frequency of term t in the document d, and $IDF(t)$ is the inverse document frequency for term t, calculated as $\log \frac{N}{1+df(t)}$, where N is the total number of documents in the collection and $df(t)$ is the number of documents containing term t.

2.1.3 Vector Space Model

In the vector space model, documents and queries are represented as vectors in the space $\mathbb{R}^{|V|}$, where V is the vocabulary of the document collection. Each dimension in this space corresponds to an indexed term (terms in the vocabulary). The weight of each dimension can be determined by different functions [22, 13], such as term frequency (TF), inverse document frequency (IDF), or a combination of these functions (TF-IDF). The similarity between a query vector and a document vector is typically calculated using the cosine of the angle between the query vector and the document vector. The cosine value is used as the relevance score for the document relative to the query. The cosine function is defined as:

$$sim(q,d) = \frac{q \cdot d}{\|q\|\|d\|} = \frac{\sum_{i=1}^{|V|} q_i \cdot d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}} \tag{2-3}$$

where $q = (q_1, q_2, \ldots, q_{|V|})$ is the vector of term weights of the query, $d = (d_1, d_2, \ldots, d_{|V|})$ is the vector of term weights of the document, and $|V|$ is the size of the collection's vocabulary (dimension of the vector space). The cosine similarity given by Eq. 2-3 generates a value between -1 and 1 for each document in the collection w.r.t the query. The

higher the similarity value, the more relevant the document is to the query. The document ranking is obtained by ordering the documents in descending order of similarity score w.r.t the query [22].

The vector space model has become the foundation of a series of IR solutions. The probabilistic IR model and the language model for IR can be seen as instantiations of the vector space model, with different term weighting schemes [22].

2.1.4 Probabilistic Models

Probabilistic models introduce probability theory as a principled basis for estimating the probability of relevance $P(r = 1|q,d)$ of a document d to a query q [22]. A document is either relevant ($r = 1$) or not relevant ($r = 0$) to a given query q. Additionally, documents and queries are represented by binary vectors, where each element of the vector indicates whether a term occurs or not. These models assume that there is a subset of all documents that the user prefers as the answer set for the query. This ideal answer set should maximize the overall probability of relevance for the user, depending only on the query and the document representations.

The Binary Independence Model (BIM) is the most original and influential probabilistic IR model [22]. It introduces the assumptions of "binary weights" and "term independence," where the latter assumes that terms occur in documents independently of each other. However, these assumptions are contrary to facts. As a result, a series of extensions has been proposed to relax some of the assumptions of the BIM, such as BM25 (BM stands for best match).

The probabilistic model BM25 is among the best weighting models for IR [39, 3, 81] and is even used in commercial systems [22]. Researchers used identifiers for variations of the formulas, and the one that achieved the best results was BM25. It extends BIM by taking into account document frequency, document length, and term frequency. The Okapi BM25 estimates the relevance of a document d w.r.t a query q as follows:

$$\text{Score}(d,q) = \sum_{t \in d,q} \log \frac{\frac{r_i+0.5}{R-r_i+0.5}}{\frac{(df_i-r_i+0.5)}{(N-df_i-R+r_i+0.5)}} \cdot \frac{(k_1+1)\cdot tf_{t_i,d}}{K+tf_{t_i,d}} \cdot \frac{(k_2+1)\cdot tf_{t_i,q}}{k_2+tf_{t_i,q}} \tag{2-4}$$

where d represents the document, q represents the query, t is a term present in both the document and the query, r_i is the number of relevant documents containing term t, R is the total number of relevant documents, df_i is the number of documents containing term t, N is the total number of documents, $tf_{t_i,d}$ is the frequency of term t in document d, $tf_{t_i,q}$ is the frequency of term t in query q, k_1 and k_2 are parameters that regulate the importance of term t in the document and query, respectively, and K is a parameter that normalizes the term frequency in the document by document length, typically defined as

$K = k_1(1-b) + b \cdot \frac{dl}{avgdl}$, where b is a parameter that regulates the impact of document length normalization, dl is the document length, and $avgdl$ is the average document length in the collection.

2.1.5 Vocabulary Mismatch Problem

Traditional sparse models suffer from the vocabulary mismatch problem [19, 15] due to semantic nuances such as synonymy and polysemy. The vocabulary mismatch problem occurs when the words used by a researcher to describe the topic of interest differ from the words used in a relevant document, as the same concept can be expressed in various ways.

Several studies have been proposed to address the vocabulary mismatch problem, and the two main lines of research are: (1) query expansion, and (2) document expansion.

Query Expansion

Query expansion is a widely used technique in IR models aimed at reducing the semantic gap between the user's query and potentially relevant documents. This technique involves adding synonyms, related terms, and other relevant terms to the original query to improve the effectiveness of the IR model. Query expansion techniques can be categorized into two main classes [75]: global, where expansion terms are selected from large document collections or lexical and semantic resources such as *WordNet*, thesauri, and ontologies; and local, where expansion terms are selected from the documents returned by the original query. The two main methods used are relevance feedback (RF) and pseudo-relevance feedback (PRF).

In general, relevance feedback uses a set of documents considered relevant by the user to reformulate the original query [16]. It requires users to identify relevant documents in the set of documents retrieved in the first stage [16, 3]. On the other hand, pseudo-relevance feedback assumes that the top k documents returned by the original query are well-representative of relevant documents, and these documents are used to reformulate the original query without user intervention [16]. The Relevance Model was one of the first approaches proposed for PRF and still maintains state-of-the-art performance [75].

Document Expansion

Document expansion, similar to query expansion, adds semantically related terms to the original document. Typically, document expansion is performed by extracting terms from external resources or the collection itself. The retrieval-based method uses the original document as a pseudo-query and performs document expansion based on the retrieved related documents [39, 3, 75, 22]. The clustering-based method allows

document expansion by clustering similar documents, adding related terms that do not occur in the document [75, 22]. Based on this, the terms of the original document and the list of additional terms for document expansion are indexed, minimizing the vocabulary mismatch problem. In general, document expansion is less explored in the literature compared to query expansion, due to the cost of regenerating the index every time the expansion technique is changed [75].

Query expansion and document expansion techniques, when applied to the sparse models, such as BM25, mitigate the vocabulary mismatch problem. However, BM25 still uses only the statistical properties of the terms contained in the documents, such as term frequency, document frequency, and document length, neglecting intrinsic properties of the documents. Moreover, traditional sparse models that adopt term weighting schemes are typically characterized as unsupervised [15, 23].

2.1.6 Learning to Rank

Traditional Ad Hoc IR models, such as BM25, perform retrieval at the level of comparison between query terms and document terms. However, there are various other statistical and intrinsic pieces of information in documents, such as the number of times a term appears in the document's title, document length, etc., which can generate important attributes that help characterize a document as more or less relevant to specific queries.

The combination of these pieces of information into new document attributes, known as feature engineering [39], involves mental effort to identify and combine information derived from documents. Feature engineering became more intensive with the advent of the web. Various pieces of information available in the HTML code of web pages are important for characterizing the importance of terms in documents, such as hyperlinks and anchor texts.

It is common to find collections where each document has more than 300 attributes. The manual development of models capable of relating hundreds of attributes to relevance values became increasingly complex and resulted in low utilization of the attributes [39, 23]. As early as the 1980s, research began to emerge proposing the use of machine learning to automatically derive models capable of better leveraging the various document attributes in relevance calculation. This gave rise to a subfield of research called Learning to Rank.

Learning to Rank reached its peak in the early 2010s with the development of models based on ensembles of decision trees. At the time, there was an emerging consensus that tree-based models, particularly gradient-boosted decision trees [83], represented the most effective solution for Learning to Rank. These models, such as LambdaMART [83], an ensemble of decision trees, combine multiple decision trees to improve ranking

effectiveness by leveraging each tree's ability to capture different aspects of queries and documents. Decision tree ensembles were implemented to solve a wide range of problems. A notable example of their success is their significant role in winning the Netflix Prize, a high-profile competition aimed at improving the quality of movie recommendations.

According to the number of documents considered in loss functions, Learning to Rank models can be grouped into three types [13, 39]:

- Pointwise approaches consider losses only on individual documents, transforming the Ad Hoc IR problem into a classification or regression problem.
- Pairwise approaches consider losses on pairs of documents and, therefore, focus on the property that A is more relevant than B.
- Listwise approaches consider losses on the entire list of documents.

However, traditional IR models fail to recognize semantic ambiguity. The complexity and diversity of languages and cultures make it difficult to accurately understand the semantic information represented in queries and documents [81].

2.2 Deep Learning Models

The deep learning approach, particularly in the field of Natural Language Processing (NLP), has made significant progress due to the increased availability of large labeled datasets and computational power, allowing researchers to employ learning methods for various purposes [23].

2.2.1 Neural Network Models

Deep learning models for IR have been used to learn vector representations (referred to as embeddings) that capture semantics through distributed modeling for queries and documents, addressing a variety of linguistic phenomena, including synonyms, paraphrases, term variations, and different expressions of similar intents [3]. These vector representations enable semantic matching, where query terms do not have to match document terms exactly in order to contribute to relevance [3, 13, 23, 39]. Consequently, neural models have become prominent in the IR community to mitigate the vocabulary mismatch problem [3, 39]. Moreover, with the use of deep learning models, the need for feature engineering is avoided, and the performance of IR models has been significantly improved, resulting in more effective systems.

Neural network models for IR can be grouped into representation-based models, interaction-based models, and hybrid models [39, 3, 13].

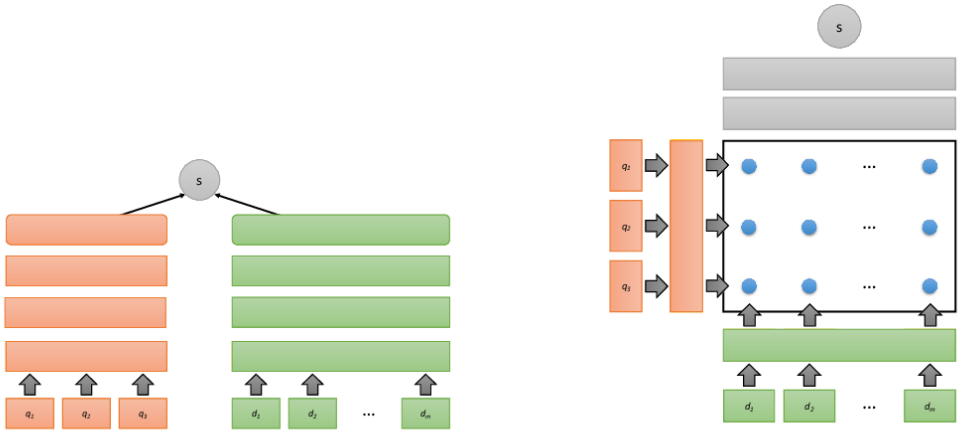


Figure 2.1: *Two classes of pre-BERT neural IR models. Representation-based models (left) learn vector representations of queries and documents, which are compared using metrics such as cosine similarity. Interaction-based models (right) explicitly model interactions between tokens in a similarity matrix. [39]*

Representation-based models aim to learn vector representations of queries and documents independently [3, 13, 81]. As a result, no interaction occurs between the query and the document during the encoding process [81]. Subsequently, metrics such as cosine similarity and inner products are used to calculate the angle between queries and documents to estimate the relevance score [13]. One of the earliest ranking models in the deep learning era is the Dual Embedding Space Model, which represents texts using pre-trained word2vec embeddings [33] and calculates relevance scores by estimating cosine similarities across all pairs of query and document tokens [39, 3, 23].

Interaction-based models capture interactions between queries and documents [39, 3, 13, 23]. Token-level similarities are usually represented within an interaction matrix where rows correspond to query tokens and columns correspond to document tokens [39, 3, 13]. These models use a similarity matrix A where each entry A_{ij} refers to the similarity between the vector of the i-th query token and the vector of the j-th document token, usually calculated in terms of cosine similarity [13, 3]. After constructing the similarity matrix, interaction-based models apply different approaches to extract features that are used to produce the query-document relevance score [13]. The DRMM and KNRM models aggregate cosine similarities between each query token and each document token, explicitly capturing interactions. KNRM uses kernel pooling for smooth matching signals to address the non-differentiability and computational inefficiency of histogram-based representation in DRMM [23]. Figure 2.1 presents a visual representation of the architectures of the two described models.

Hybrid models combine the architecture of representation-based and interaction-based models. For instance, the DUET model comprises two parallel models: a local model, which focuses on exact matching and learning interaction signals between document and query inputs; and a distributed model, which relies on distributed representations of the input for semantic matching [3, 23]. The relevance score is obtained by summing the scores from the local and distributed models.

Although all these approaches improve performance across a variety of IR systems, they do not consistently perform well when trained on small-scale data [81]. Additionally, models like word2vec, widely used in pre-BERT models [39, 3], do not adequately capture contextual information.

2.2.2 Pre-trained Language Models

In a broad sense, according to Wang et al.[81], a pre-trained language model (PLM) refers to a language model trained on large-scale data, including models based on static vector representations (e.g., word2vec [33]) and models based on dynamic contextualized vector representations (e.g., ELMo [62], GPT [64], and BERT [12]). Improvements in vector representations have enabled the modeling of tokens within a specific textual context using models such as ELMo [62] and BERT [12]. The idea is to first pre-train the model on the self-supervised masked language modeling (MLM) task by predicting the probability distribution over the vocabulary of a masked token, and then adapt the pre-trained model to specific downstream tasks by fine-tuning the parameters with task-specific objectives [13].

Unlike earlier neural IR models, known as pre-BERT models, contextualized vector representations address the semantic mismatch problem, where the same token can refer to different concepts. Transformer-based ranking models [80], such as GPT, BERT, and their derivatives, offer significant performance improvements by mapping queries and documents to contextualized representations using attention mechanisms that focus on critical parts and capture the intrinsic semantics of the query and the document.

Transformer-based models have revolutionized the IR field and are the state-of-the-art in Ad Hoc IR. Among the various encoder architectures, BERT [12] stands out. The first successful use of BERT in Ad Hoc IR was the method known as monoBERT [59], marking the beginning of the BERT revolution in IR [39]. For this reason, Section 2.3 presents the architecture of BERT.

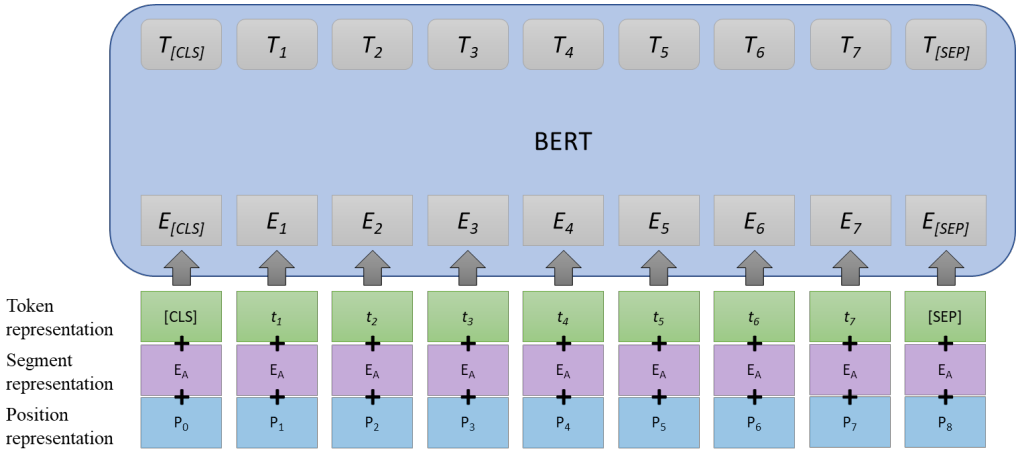


Figure 2.2: *General architecture of BERT. The input vectors are the sum of token, segment, and position embeddings. [39]*

2.3 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language model designed to learn bidirectional vector representations of text. Unlike unidirectional language models such as GPT [64], BERT can consider context from both the left and right of a given token [12].

Regarding encoding, the input (sequence of tokens) is converted into token embeddings (or vectors), learned during training, which are added with positional and segmental embeddings to provide the model information about the position of each token in the sequence and to encode which of the two input sentences a token belongs to, respectively. The model generates a corresponding sequence of contextualized vector representations for the input, where the vector of a token is influenced by the context in which it appears. The notation for the input (E) and output (T) of BERT is denoted as:

$$[E_{[CLS]}, E_1, E_2, \ldots, E_n, E_{[SEP]}] \tag{2-5}$$

$$[T_{[CLS]}, T_1, T_2, \ldots, T_n, T_{[SEP]}] \tag{2-6}$$

where E_i, $1 \leq i \leq n$, are the vectors of the tokens in the original text tokens. The [CLS] and [SEP] vectors are special tokens added at the beginning and end of each input sequence, respectively. When the task involves two inputs, [SEP] indicates whether a token belongs to input A or input B. T_i, $1 \leq i \leq n$, are the contextualized vector representations of the corresponding tokens.

The token vectors correspond to the subwords of the BERT vocabulary, which are learned during training. For instance, BERT uses the WordPiece tokenizer [84], capable

of modeling large collections with millions of unique words while maintaining a reduced vocabulary of approximately 30,000 tokens [3, 39]. This reduced vocabulary is achieved by splitting words into "subwords." For example, the word " walking" might be split into "walk" and "##ing," where "##" indicates that the current token is connected to the previous subword. Note that the splitting process is generally unsupervised, and the resulting subwords are not necessarily linguistically meaningful [39].

The input vectors of BERT are initially processed through a self-attention mechanism with 8 attention heads (multi-head self-attention). Each token is represented by three vectors per head: query, key, and value [16]. These vectors are obtained through linear projections, which convert the initial vectors (with 512 dimensions) into multiple 64-dimensional vectors. Self-attention for the input is calculated in each head, as shown in Eq. 2-7, based on Vaswani et al.[80].

$$Attention(Q,K,V) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V, \tag{2-7}$$

where d_k is the dimension of the key vector.

The outputs of the 8 heads are concatenated (8 * 64 dimensions) and projected back to the original 512 dimensions. This process is followed by a residual connection and layer normalization [16], which are also applied to the feed-forward neural network (FFNN) to obtain the final contextualized vector representation.

2.3.1 Pre-training

BERT [12] introduced two self-supervised pre-training objectives: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).

Masked Language Modeling

During pre-training, approximately 15% of the tokens in the training input are randomly masked, and the model is optimized to predict these masked tokens using the context on both the left and right sides (bidirectionally) through cross-entropy. This masking is performed as follows: in 80% of the time, the tokens are replaced with the special token [MASK]; in 10% of the time, they are replaced by random words; and in the remaining 10% of the time, the original words are kept.

This procedure helps mitigate the mismatch (gap) between pre-training and fine-tuning, as the [MASK] token does not appear during fine-tuning [12, 39, 3, 16]. For MLM task, the inputs pass through all layers of BERT. At the end, a probability distribution over the vocabulary is used to predict which words was replaced by the masked tokens. Figure 2.3 provides a visual representation of BERT's masked language modeling.

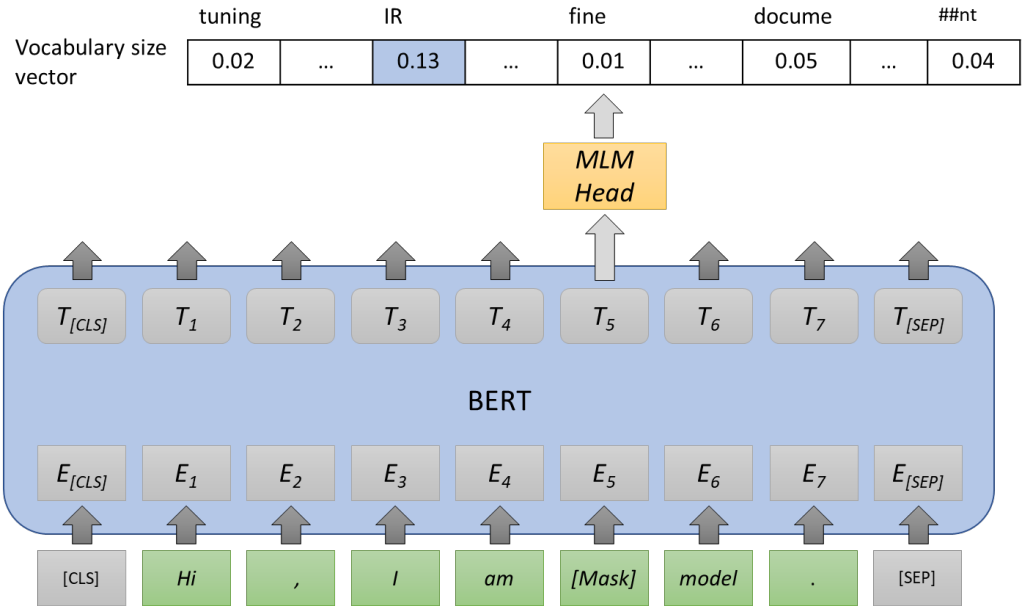


Figure 2.3: *BERT's MLM. In this example, the word "model" is masked in the input sequence and replaced with the special token [MASK]. The final hidden vectors are processed through an MLM head over the vocabulary. The output vector contains the probability that the masked token corresponds to the i-th token in the vocabulary.*

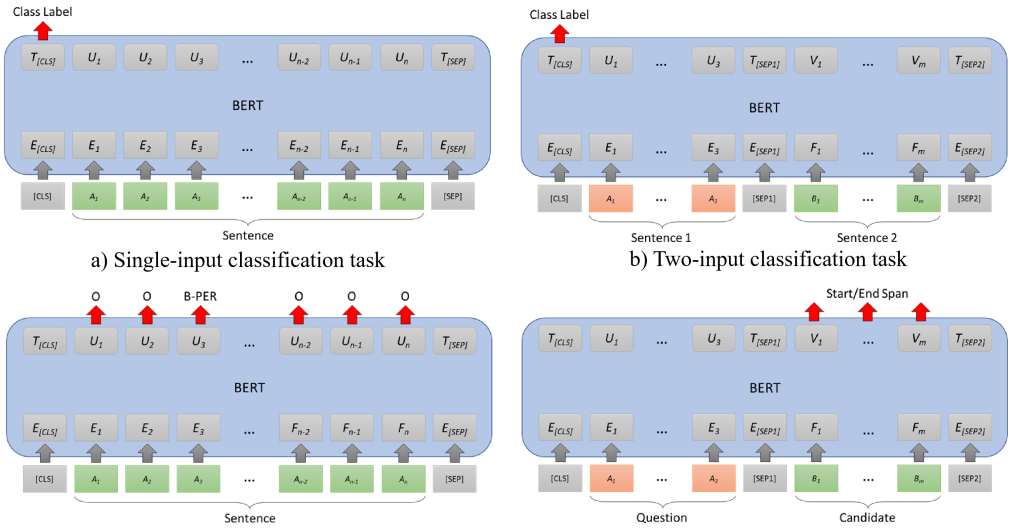
Next Sentence Prediction

Given that many downstream tasks, such as question answering (QA) and natural language inference (NLI), rely on understanding the relationship between two sentences, BERT is pre-trained on pairs of sentences separated by the [SEP] token. It attempts to predict whether the second sentence follows the first in a collection, where 50% of the time the second sentence indeed follows the first. The Next Sentence Prediction task uses the [CLS] output token of BERT to estimate the probability that the second sentence follows the first. However, the actual necessity of this pre-training objective has been questioned in the literature.

2.3.2 Fine-Tuning

In fine-tuning for specific tasks, one or more fully connected layers are typically added to the last layer of BERT. Devlin et al.[12] proposed four models for specific tasks, illustrated in Figure 2.4, by adding a single additional output layer to BERT, which is trained from scratch during fine-tuning [12, 39, 3]:

- Single-input classification task: Used to assign a label to a sequence of tokens, such as in sentiment analysis or other similar tasks.



a) Single-input classification task
b) Two-input classification task
c) Single-input token labeling task
d) Two-input token labeling task

Figure 2.4: *Illustration of how BERT is used for different natural language processing tasks. [39]*

- Two-input classification task: For example, identifying whether two sentences are paraphrases or semantically similar to each other.
- Single-input token labeling task: Used to assign labels to each token in a sequence of tokens, such as in named entity recognition.
- Two-input token labeling task: Used to mark the start and end positions of the answer span within a candidate paragraph (the second input), given a question (the first input), for example, in question answering.

The original BERT paper introduced two model variations: $\text{BERT}_{\text{base}}$ (layers=12, hidden dimensions=768, self-attention heads=12, parameters=110M) and $\text{BERT}_{\text{large}}$ (layers=24, hidden dimensions=1024, self-attention heads=16, parameters=340M). Due to its more complex architecture and larger number of hyperparameters, $\text{BERT}_{\text{large}}$ outperforms $\text{BERT}_{\text{base}}$ under the same training and fine-tuning procedures [12, 81]. Due to the quadratic time and memory complexity of the self-attention mechanism in the standard transformer, the input size of transformer-based models is limited to 512 tokens (including the [CLS] and [SEP] tokens).

The special input token [CLS] is added at the beginning of the input sequence, and its final hidden state, T[CLS], is used as input to the additional layer for classification tasks. T[CLS] is considered as an aggregated vector of the entire input (whether single or double) [3]. In IR models, the [CLS] token is often selected as a vector for queries and documents and optimized for the IR task. However, other approaches use all tokens, establishing BERT as a basis for IR. The following subsections will discuss the application of BERT-based models to generate suitable vectors for ranking in a supervised scenario,

divided into dense models 2.4, sparse models 2.5, and hybrid models 2.6.

2.4 Dense Models

Traditional and pre-BERT IR models face challenges related to vocabulary mismatch and semantic ambiguity, respectively. Despite efforts to mitigate these issues through query and/or document expansion before retrieval, aiming to bridge the vocabulary mismatch gap, and the use of static token representations to address semantic ambiguity, these approaches still present significant limitations.

Static token vectors cannot model polysemy, as the usage of these tokens varies across linguistic contexts—that is, the same token can refer to different concepts. Recent studies [62, 64, 12] have proposed contextualized vector representations that address the problem of semantic mismatch. BERT-based ranking models offer significant performance improvements by generating contextualized representations for queries and documents, enhancing the model's ability to overcome vocabulary and semantic mismatches. This shift has undeniably promoted the transition from sparse signals, mostly limited to exact matches, to contextual vectors capable of capturing semantic matches and nuances between tokens, thereby improving model effectiveness.

This paradigm shift, along with the first successful application of BERT proposed by Nogueira and Cho [59], sparked a new wave of IR models that perform retrieval in two main ways: in the second stage, with cross-encoder models (discussed in more detail in Section 2.4.1), where the query and document are processed together through an encoder that performs full attention over all input tokens; and in the first stage, with bi-encoder models (discussed in more detail in Section 2.4.2), where twin networks (from which the name bi-encoder or dual encoder derives) receive queries and documents and encode them independently into contextualized vectors, then the relevance score is calculated as the similarity between the query and document vectors.

2.4.1 Cross-Encoder Models

The first proposed BERT-based ranking model, known as monoBERT [59], applies probability theory as a foundational principle for estimating the relevance probability of a document d w.r.t a query q. The probability principle states that documents should be ranked in descending order of the estimated relevance probability with respect to the information need [22, 39, 3]. The model achieved state-of-the-art performance on the MS MARCO passage retrieval task, outperforming the traditional BM25 model by 117% in MRR@10. In this approach, the IR task is transformed into a binary relevance classifica-

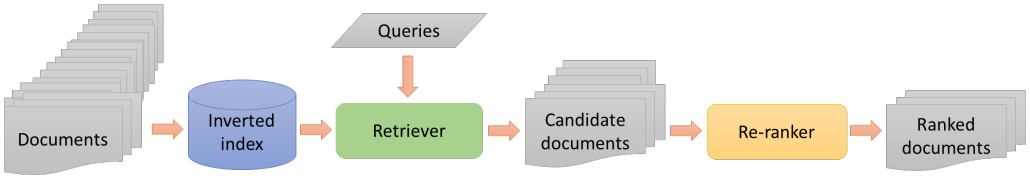


Figure 2.5: *Multi-stage architecture. In the first stage, candidate documents are retrieved by a sparse model, typically BM25, and in the second stage, BERT inference is applied to re-rank the candidate documents.*

tion task, where a score (the probability) is estimated for a document d belonging to the "relevant" class for a query q, denoted as:

$$P(\text{relevant} = 1 | d_i, q) \tag{2-8}$$

Cross-encoder models can be abstracted as:

$$\text{rel}(q,d) = f(\eta(q,d)) \tag{2-9}$$

where η is the BERT encoder that takes the query q and document d as input, similar to the two-input classification task explained in Subsection 2.3.2. f is the relevance function that estimates the score according to the probability of the document being relevant to the query. The function f is implemented by an additional neural network layer with a softmax output, as shown in Eq. 2-12.

In this way, the interaction between query and document tokens is modeled within the encoder using the self-attention mechanism. Note that the interaction cannot be pre-computed until the query arrives, which implies that it is more efficient to use these models for re-ranking a small set of documents due to the high computational cost of processing all query-document pairs in the data collection [13]. Therefore, the cross-encoder model is used as a re-ranker in a multi-stage IR architecture, as illustrated in Figure 2.5.

Most BERT applications are based on multi-stage IR architectures, where a sparse model, such as BM25, generates a list of candidate documents, and then BERT inference is applied to re-rank these candidates, producing a score for each document d_i in the list. The scores derived from BERT may or may not be combined or aggregated with other relevance signals to arrive at the final scores used for re-ranking [39]. Nogueira and Cho [59] used BERT scores directly to re-rank the candidates. Other approaches leverage the scores from the sparse model in the initial retrieval, known as hybrid models, explained in Section 2.6.

In the re-ranking phase, cross-encoder models are widely used and more effective than bi-encoder models, described in Section 2.4.2. However, bi-encoder models are more

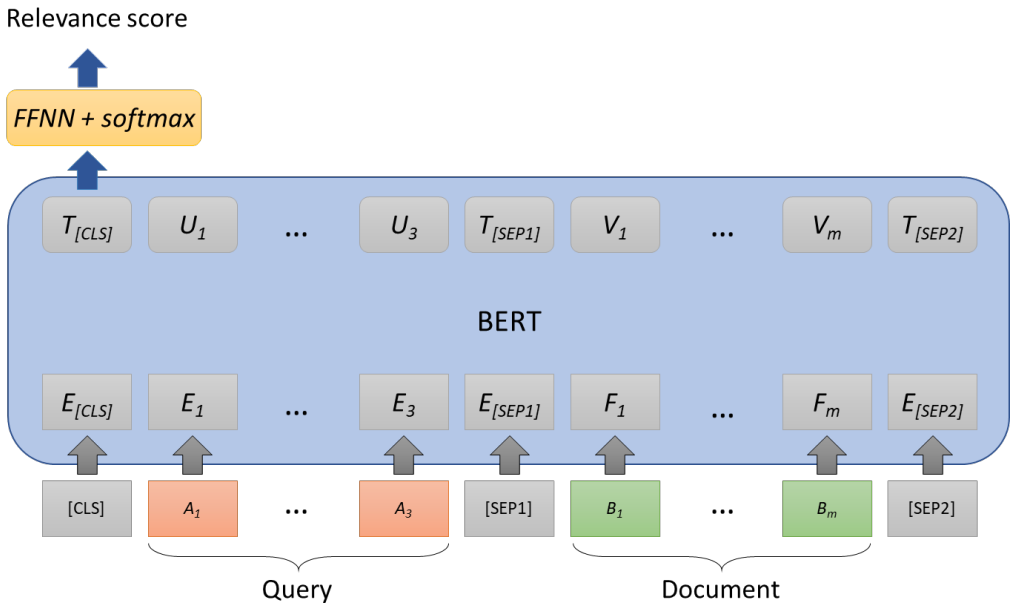


Figure 2.6: *Architecture of monoBERT. [39]*

efficient, as they can pre-compute document vectors to reduce online inference time.

Architecture of monoBERT

The architecture of monoBERT is illustrated in Figure 2.6. The notation for the input (E) and output (T) of monoBERT is denoted as:

$$[E_{[CLS]}, q, E_{[SEP1]}, d_i, E_{[SEP2]}] \tag{2-10}$$

$$[T_{[CLS]}, cq, T_{[SEP1]}, cd_i, T_{[SEP2]}] \tag{2-11}$$

In Eq. 2-10, q corresponds to the sequence of query token vectors. d_i corresponds to the sequence of candidate document token vectors d_i. This is the same input sequence configuration as in Figure 2.4 (b) for classification tasks involving two inputs. In Eq. 2-11, cq corresponds to the sequence of contextual vectors of the query tokens generated by BERT, and cd_i corresponds to the sequence of contextualized vectors of the terms of document d_i generated by BERT.

monoBERT uses the concatenation of the query q and the document d, including the special token [CLS] at the beginning and two special delimiter tokens [SEP], placed at the beginning and end of the document segment as input. It feeds the BERT [CLS] output vector ($T_{[CLS]}$) to a feed-forward neural network to obtain the relevance score of the given query and document, as shown in Eq. 2-12.

$$\text{rel}(q, d_i) = P(\text{Relevant} = 1 | q, d_i) = \text{softmax}(T_{[CLS]} \cdot W + b)_1 \tag{2-12}$$

where $T_{[CLS]} \in \mathbb{R}^D$, D is the dimensionality of the model's contextualized vector, $W \in \mathbb{R}^{D\times 2}$ is a weight matrix of the feed-forward neural network, $b \in \mathbb{R}^2$ is a bias, and $\text{softmax}(\cdot)_i$ with two dimensions denotes the i-th element of the softmax output over all possibilities, i.e., "relevant" and "not relevant."

The fully connected neural network, randomly initialized, is added on top of the pretrained BERT encoder. This network is then trained end-to-end along with the BERT layers for the relevance classification task using the cross-entropy loss function, presented in Eq. 2-13.

$$L = -\sum_{j\in J_{pos}} \log \text{rel}(q,d_j) - \sum_{j\in J_{neg}} \log(1-\text{rel}(q,d_j)) \tag{2-13}$$

where J_{pos} is the set of indices of relevant documents and J_{neg} is the set of indices of non-relevant documents. Since the loss function considers only one document at a time, it can be characterized as belonging to the pointwise approach family of learning to rank, presented in Section 2.1.

2.4.2 Bi-encoder Models

One of the bottlenecks limiting the efficiency of cross-encoder models arises from BERT's self-attention mechanism. During inference, a query must pass through BERT layers simultaneously with each document from the collection[1]. This approach makes the use of cross-encoders impractical as a retriever (first-stage ranker) due to the high computational cost, resulting in an unacceptable response time.

Shortly after the emergence of monoBERT, researchers began investigating approaches to decouple query processing from document processing. Given that a document collection can be extremely large, the ideal approach is to pre-compute and store document vectors, similar to what is done in pre-BERT representation-based models, where representations generated during the indexing phase are later used in query processing.

Nevertheless, to achieve a better balance between effectiveness and efficiency, recent studies have employed a bi-encoder architecture [27], consisting of two encoders to independently learn vectors representations for queries and documents.

Given a collection $\mathcal{C}$ of documents and a query q, the task is to generate a ranking of documents from the collection $\mathcal{C}$ w.r.t q aiming to optimize some similarity measure.

The goal is to learn (train) a transformation $\eta : [t_1,\ldots,t_n] \implies \mathbb{R}^n$ for documents and queries from the collection. Let $\eta_q(\cdot)$ denotes the transformation for the query and $\eta_d(\cdot)$ denotes the transformation for a document d in $\mathcal{C}$. The functions $\eta_q(\cdot)$ and $\eta_d(\cdot)$ are implemented by an encoder (BERT or another pre-trained language model) and may be the same encoder or two distinct encoders. They correspond to a query encoder and

[1] Or the set of documents retrieved by a first-stage retriever in case of multi-stage IR architectures

a document encoder, respectively. A similarity function $\phi(.)$ is trained together with the encoders $\eta_q(\cdot)$ and $\eta_d(\cdot)$ to maximize similarity $\phi(\eta_q(\cdot),\eta_d(\cdot))$ for documents d relevant to q and minimize similarity $\phi(\eta_q(\cdot),\eta_d(\cdot))$ for documents d not relevant to q. The function $\phi(\cdot)$ is usually implemented as the dot product or the cosine similarity between the query vector and the document vector.

Note that the encoders $\eta_q(\cdot)$ and $\eta_d(\cdot)$ are theoretically distinct, meaning that document vector generation does not depend on queries. Therefore, since the documents are known, the document encoder $\eta_d(\cdot)$ can be applied to the entire document collection to generate the contextualized vector[2] for each document and index them offline. This process facilitates query processing and significantly reduces latency.

During online inference, the query encoder $\eta_q(\cdot)$ generates the contextualized vector of the query. Then, the similarity function $\phi(\cdot)$ is used to compute relevance scores based on the similarity between the query vector and the pre-computed document vectors. Thus, the relevance of a document d_i from a collection C w.r.t a given query q is formulated as follows [39, 3, 13]:

$$rel(q,d) = \phi(\eta_q(q),\eta_d(d)) \tag{2-14}$$

where η_q and η_d denote encoders for the query and document, respectively. ϕ is a similarity function.

Unlike cross-encoder models, bi-encoder models can be easily applied for first-stage retrieval in large collections, using efficient nearest neighbor search algorithms, presented in the following subsection. Bi-encoder models are categorized into two families (illustrated in Figure 2.7): single-vector models, where the entire input is represented by a single dense vector, i.e., by the output contextualized vector $T_{[CLS]}$ from the encoders; and multi-vector models, which relax the constraints of single-vector models by including multiple token vectors. Both families of bi-encoders will be discussed in their corresponding subsections later in this section.

Nearest Neighbor Search

The ranking procedure for bi-encoder models can be either multi-stage (similar to monoBERT) or single-stage (end-to-end). Due to the ability to pre-compute vectors, bi-encoder models are generally applied in a single stage. The single-stage ranking procedure can be described in steps, explained in more detail in Sections 2.8.3, 2.8.4, and 2.8.5, where the first stage involves retrieving tokens similar to the query tokens through a

[2]Or vectors, depending on whether the bi-encoder is single-vector or multi-vector, as will be explained later in this section.

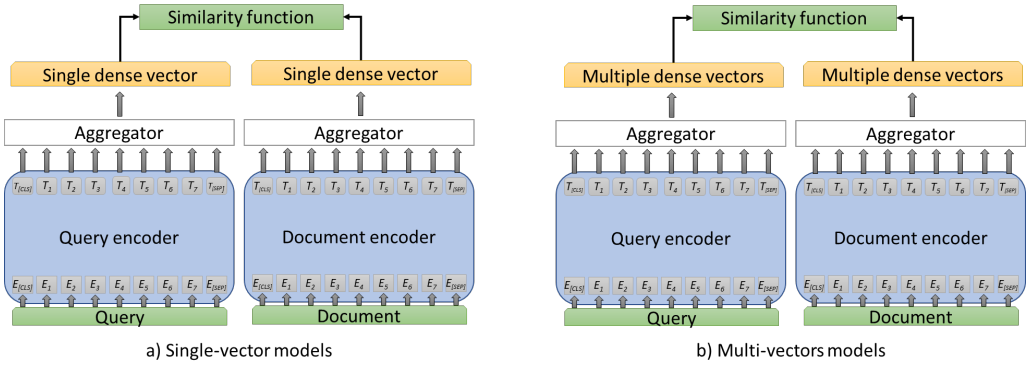


Figure 2.7: *Two families of bi-encoder models. a) Single-vector models compute vectors for the query and/or document and aggregate them into a single dense vector. b) Multi-vector models compute vectors for the query and/or document and aggregate them into multiple dense vectors.*

nearest neighbor search (NNS), and the final stage calculates the similarity between the query and the documents.

For indexing the vectors, an infrastructure that supports document retrieval is required. A solution for this is a vector index or a vector database that supports nearest neighbor search, such as the Faiss library [28] from Facebook, widely used in the IR community for bi-encoder models.

The nearest neighbor search problem is formulated as follows: given a collection of documents $\mathcal{C}$ and a query q, the goal is to find a list of k nearest documents to the query q, denoted by S. This search can be performed based on different comparison functions, typically using the dot product or Euclidean distance. Obviously, there is a trivial solution to this task, called "brute force" [39, 3]: assuming it is possible to calculate the similarity between any two vectors, we can compute the similarity between the query q and the documents in the collection $\mathcal{C}$, store the results in a list, and extract the documents with the maximum similarity. This procedure requires storing the $|\mathcal{C}|$ document vectors and performing $|\mathcal{C}|$ similarity calculations.

In Ad Hoc IR tasks, the collections $\mathcal{C}$ are usually very large (often exceeding one billion documents), making the brute-force approach for nearest neighbor search quickly impractical as the collection size grows [82, 39, 3]. Therefore, it is undesirable to traverse the entire collection for each query. Instead, it is expected to preprocess $\mathcal{C}$ into some data structure that allows answering each query in substantially sublinear time. Ideally, the query time should be logarithmic or polylogarithmic in $|\mathcal{C}|$, while keeping the data structure small. Thus, exact NNS cannot meet the real-world efficiency and cost requirements. As a result, much of the literature has focused on efforts to search for approximate nearest neighbors (ANNS) and find algorithms that substantially improve

efficiency while slightly relaxing precision constraints (a trade-off between precision and efficiency) [82].

The approximate nearest neighbor search problem is formulated as follows: given a vector space V of dimension d, and a document collection $\mathcal{C}$ containing n vectors in this space, the $(1+\varepsilon)$-ANN problem consists of, given a query q, finding a vector a in the collection such that the distance between q and a is at most $1+\varepsilon$ times the distance between q and any other vector b in $\mathcal{C}$. The parameter $\varepsilon > 0$ represents the slackness allowed in the approximation. The goal is to efficiently find approximate nearest neighbors, especially when the dimension d of the vector space is high.

According to the adopted index, existing ANNS algorithms can be divided into four main categories [4]: hashing-based, tree-based, quantization-based, and graph-based algorithms [82, 3]. Recently, graph-based algorithms have emerged as a highly effective option for approximate nearest neighbor search due to their extraordinary ability to express neighborhood relationships. These algorithms need to evaluate fewer data points to obtain more accurate results [82].

Approaches based on hierarchical navigable small world (HNSW) graphs represent the current state-of-the-art in approximate nearest neighbor search, as indicated by a popular benchmark [39, 3]. Graph-based methods construct the index by retaining information about the neighborhood of each data point toward other data points or sets of articulation points. Therefore, several greedy heuristics are employed to navigate the proximity graph w.r.t a given query point.

Single-Vector Bi-Encoder Models

The first bi-encoder architecture employing BERT emerged to solve the problem of semantic textual similarity between two sentences in the NLP community. The semantic similarity task differs from the IR task in several aspects, such as the size of the two inputs, linguistic structure, and spelling errors. Typically, in IR tasks, queries are short compared to documents and tend to be more exposed to spelling errors and inadequate linguistic structure. This contrasts with the semantic similarity task, which exhibits richer linguistic structure and error-free writing. Additionally, the lengths of sentences A and B are equitable, without significant discrepancies.

Reimers and Gurevych [68] introduced Sentence-BERT, an adaptation of BERT to learn how to generate vectors that allow the calculation of semantic similarity between two sentences (or between two vectors). This model can be described as the first model based on a bi-encoder architecture, a term introduced by Humeau et al.[27]. Sentence-BERT, despite being a bi-encoder, uses the same encoder for both sentences because the semantic textual similarity task is symmetric.

The authors used BERT as the base encoder and propose three pooling strategies to produce a single vector for the sentence:

- Only the output of special token [CLS], that is, the contextualized token vector T[CLS].
- Mean pooling over all contextualized vectors of BERT output.
- Max pooling over all contextualized vectors of BERT output.

During training, a single-layer neural network is added on top of BERT and receives as input a single vector generated through one of the three pooling strategies. The entire model is trained end-to-end. The authors report that the pooling strategy and objective function (classification, regression, or triplet) depend on the available training dataset.

At inference time, the trained encoder is applied to both sentences, producing sentence vectors u and v. The cosine similarity between these two vectors is directly interpreted as a similarity score. However, Reimers and Gurevych [68] do not address the IR problem.

The application of the bi-encoder architecture using BERT to the IR problem, to the best of our knowledge, was introduced by Lee et al.[35] for open-domain question answering (OpenQA). The single-vector model was pre-trained using the unsupervised Inverse Cloze Task. However, recent studies [29, 82] suggest that pre-training using the Inverse Cloze Task does not seem to be as effective as supervised single-vector models. Therefore, supervised models from the NLP (Dense Passage Retrieval - DPR [29]) and IR (Approximate Nearest Neighbor Negative Contrastive Learning - ANCE [82]) communities have emerged as promising approaches for dense retrieval.

DPR [29] is proposed for OpenQA tasks. The model learns vectors for queries and passages using two independent BERT-based encoders, both of which use only the output vector corresponding to the [CLS] token. Relevance scores are then calculated using the dot product between the query vector and the document vectors. Results on various OpenQA datasets show that DPR outperforms BM25.

For Ad Hoc retrieval tasks, Zhan et al.[88] proposed that RepBERT replace BM25 in the retrieval component. The RepBERT model architecture is similar to DPR, except that RepBERT uses mean pooling of the vectors and a shared BERT-based encoder for queries and documents.

To train a ranking model, a triplet (discussed in more detail in Section 2.8.4) composed of a query, a relevant document, and a non-relevant document is required. Non-relevant documents are primarily obtained through techniques such as random selection, top BM25 candidates, and in-batch negative selection, with the latter often producing uninformative negative examples, similar to random selection.

ANCE [82] adopts the same architecture as DPR. However, like RepBERT, ANCE opts for the same encoder to independently generate query and document vectors. Additionally, the authors observed that selecting negative documents from BM25 results may not be the best strategy, as it tends to bias the model to mimic the sparse model. Therefore, they proposed using the model itself to perform an approximate search for the nearest negative documents to the query.

During training, in the first phase, the single-vector model is warmed up with BM25 negatives. In the second phase, hard negatives are selected via ANNS on an index of current document vectors generated by the encoder being trained. The index is updated periodically, asynchronously, and new vectors for the entire collection are computed. This process typically occurs every m batches. Hard negatives are documents (passages) whose vectors have high similarity with the query vector but are not relevant documents. Experimental results indicate that ANCE elevates single-vector models and convincingly outperforms baselines on several benchmarks [82, 13].

Zhan et al.[87] proposed a new technique for training single-vector models, called STAR (Stable Training Algorithm for dense Retrieval) + ADORE (Algorithm for Directly Optimizing Ranking pErformance). STAR follows a training procedure similar to ANCE. The document encoder weights are frozen after the STAR training procedure, and document vectors are pre-computed. The document index is permanently built from the vectors generated by the document encoder. Then, ADORE, a query-side training algorithm, trains the query encoder. In each training iteration, the algorithm encodes a batch of queries and uses these vectors to retrieve the most relevant corresponding negative documents in the document index, which act as dynamic hard negatives used to train the query encoder. Experimental results on passage and document ranking tasks show that the proposed method significantly outperforms all competitive sparse and dense retrieval models [87].

Single-vector models seem to be an intermediate solution between sparse models and cross-encoder models. They demonstrate greater effectiveness than BM25 but are less efficient than BM25 in both time and space. Single-vector models are less effective than cross-encoder models but are significantly more efficient [39, 3]. Additionally, they enable direct ranking of the entire collection using pre-computed document vectors. Finally, there appears to be synergy between sparse and dense retrieval, in the sense that their combination can improve the effectiveness of each individually, as discussed in Section 2.6.

Multi-Vector Bi-Encoder Models

The primary motivation behind the bi-encoder architecture is to sacrifice a fraction of the high effectiveness of cross-encoders to gain efficiency, thereby enabling

first-stage retrieval in a document collection.

Driven by this challenge, researchers began exploring different trade-off solutions between effectiveness and efficiency, relaxing the constraints of single-vector models. Recently, several approaches have been proposed to improve the efficiency and effectiveness of BERT-based models for IR. Zhang et al.[58] introduced DC-BERT (Decoupled Contextual Encoding Framework), which uses dual BERT encoders to optimize the interaction between queries and documents. This method pre-computes documents offline, storing the contextualized vectors of the tokens, while the query is encoded online only once. During inference, the contextualized token vectors are integrated using a high-layer Transformer initialized by the last k layers of the pre-trained BERT [22]. The number of layers k is configurable for a trade-off between model effectiveness and efficiency.

On the other hand, Cao et al.[6] and MacAvaney et al.[49] proposed DeFormer (Decomposed Transformer) and PreTTR (Precomputing Transformer Term Representations) to decompose the lower layers of BERT, replacing full self-attention with self-attention on the entire query and document. PreTTR also introduced a compression layer to reduce the required storage without substantial degradation in retrieval performance [49].

The PreTTR model stood out by decoupling the encoding of queries and documents, enabling significant acceleration in TREC WebTrack. However, the effectiveness of this strategy may vary depending on the specific dataset.

MORES (MOdularized REranking System), introduced by Gao et al.[18], adopted a similar approach by modularizing the Transformer-based re-ranker (such as monoBERT) into separate representation and interaction modules. One of the key differences is that the interaction module in MORES is not fully a cross-attention mechanism; only query-to-document attention is executed, followed by query self-attention [18, 13]. In this way, document vectors remain unchanged for all queries. In their experiments, the authors demonstrated that two lightweight interaction modules can compete in ranking performance with more complex architectures, achieving remarkable acceleration.

However, most of the computational cost comes from the interaction of the last layers used in these models. Therefore, an alternative for multi-vector models that generate vectors and capture explicit token-level interactions are the Poly-encoder, ME-BERT, and ColBERT models, presented in Sections 2.8.1, 2.8.2, and 2.8.3, respectively. These models employ an interaction function with lower computational cost but high effectiveness.

2.5 Sparse Models

Sparse models refer to methods that represent documents and queries in such a way that most values in the vector are zero. A classic example of a sparse model is BM25, which uses a vector representation based on term frequency and inverse document frequency (TF-IDF). In this approach, each document and query are represented by a vector where each dimension corresponds to a term in the vocabulary, and the values represent the frequency or relevance of that term in the document or query.

Queries typically have few terms and tend to be sparser than documents. This characteristic of sparse models makes them very efficient in query processing and effective at capturing exact term matches. The efficiency comes from the use of inverted indexes, which are structures adapted to the high sparsity of terms in documents[3]. However, sparse models may fail to capture semantic matches (synonyms, contextual relationships, or similar ideas) when exact terms do not match between documents and queries.

BM25 does not handle the vocabulary mismatch problem well. To address this issue, several solutions have been proposed. In general, Section 2.1 introduced two main lines of research that address the vocabulary mismatch problem: (1) query expansion and (2) document expansion.

Query expansion is a widely used technique in IR models that involves adding synonyms, related terms, and other relevant terms to the original query to improve the effectiveness of the IR model.

Naseri et al.[54] proposed a model of contextualized embeddings for query expansion (CEQE) that uses dense contextualizd vectors from BERT to calculate the probability $p(\cdot)$ of a candidate expansion token w given a query q and a document d extracted from the top-k documents retrieved by the original query. The CEQE model is based on the following Eq. 2-15:

$$\sum_{d \in R} p(w,q,d) = \sum_{d \in R} p(w|q,d)p(q|d)p(d) \tag{2-15}$$

where $p(w,q,d)$ is the joint probability of a candidate expansion token w, a query q, and a document d. R denotes the top-k documents retrieved by the original query. The joint probability $p(w,q,d)$ is decomposed using the chain rule of conditional probabilities. $p(w|q,d)$ is the probability of occurrence of the candidate token w given a query q and a document d. $p(q|d)$ is the probability of occurrence of the query q given a document d. $p(d)$ is the probability of occurrence of the document d.

The CEQE model is used as a retriever in the first stage of the multi-stage IR architecture, aiming to generate an initial list of candidate documents (with higher recall)

[3] Most terms occur in few documents, according to Zipf's Law [91].

that will later be ranked by a re-ranking model in the second stage.

The authors argue that the use of contextualized vectors from BERT allows capturing semantic relationships between tokens more effectively than methods based solely on co-occurrence. Experimental results on two TREC datasets, Robust and Deep Learning, demonstrated significant improvements in MAP and nDCG metrics compared to traditional query expansion methods.

Another less explored line of research is document expansion. Nogueira et al.[60] proposed the first successful application of BERT for document expansion, introducing the doc2query technique. Unlike query expansion, document expansion adds semantically related tokens to the original document, aiming to improve the match between the document and potential user queries.

In the doc2query model, the authors trained a transformer model on the MS MARCO passage collection for a total of 30 epochs. During indexing, given an input document d_i, the doc2query model is used to generate a list of artificial queries related to the document d_i. The generated queries are then appended sequentially to the end of the original document d_i, without any markers to distinguish them from the original content.

The authors observed that the doc2query model tends to copy some words from the input document, meaning it can effectively increase the importance of key tokens present in the original document. Additionally, doc2query also produces tokens not present in the input document, which can be characterized as an expansion through synonyms and other semantically related tokens.

The experimental results presented in the work of Nogueira et al.[60] demonstrated that the doc2query approach achieved significant gains of 15% in MRR@10 on the TREC CAR and MS MARCO datasets compared to the BM25 baseline.

2.6 Hybrid Models

Hybrid models typically combine traditional statistical-based methods (such as BM25) with more recent techniques that use document vectors (such as BERT-based models). These models leverage the strengths of each approach to compensate for their limitations.

A study conducted by Ma et al.[47] demonstrates that a dense-sparse (linear) hybrid combination of DPR with BM25 results in significant gains over dense or sparse retrieval alone. Additionally, Luan et al.[44] developed a version of ME-BERT (discussed in Section 2.8.2), which combines dense retrieval results with BM25's sparse retrieval through a linear combination of scores to create a more effective dense-sparse hybrid model, confirming the potential of hybrid models compared to dense or sparse retrieval alone.

Given a query q and a document d, the final score is generally calculated using Eq. 2-16:

$$\text{score}_{final}(q,d) = BM25(q,d) + \lambda \cdot sim(\eta_q(q), \eta_d(d)) \tag{2-16}$$

where *sim* is a function that calculates the similarity between two dense vectors obtained by two encoders $\eta_q(q)$ and $\eta_d(d)$ for the query q and document d, respectively. BM25 represents the lexical model. λ is an adjustable parameter determined through cross-validation.

Gao et al.[20] proposed the Complementary Lexical Retrieval Model with Semantic Residual Embeddings (CLEAR), which uses a bi-encoder to complement the sparse model (BM25). CLEAR effectively trains a hybrid model where BM25 scores and bi-encoder scores contribute to adjusting the weights of the BERT in the bi-encoder. The model uses only one BERT; to differentiate inputs, instead of using the [CLS] token, special tokens [QRY] and [DOC] are added at the beginning of queries and documents, respectively.

The final dense vector is obtained through mean pooling, generating a dense vector $\mathbf{v_d}$ for the document and a single dense vector $\mathbf{v_q}$ for the query. The similarity score is calculated as the dot product between these vectors. The final score is obtained by combining the BM25 score *BM25* and the dense model score *sim*, as shown in Eq. 2-17.

$$\text{score}_{final}(q,d) = \lambda \cdot BM25(q,d) + sim(\eta_q(q), \eta_d(d)) \tag{2-17}$$

where λ regulates the impact of the two scores. $\text{sim}(\cdot)$ is a function that calculates the similarity between two dense vectors obtained by two encoders $\eta_q(q)$ and $\eta_d(d)$ for the query q and document d, respectively.

2.7 Evaluation of Ad Hoc IR Systems

An Ad Hoc IR system can be evaluated from different perspectives. However, the most important aspects are effectiveness[4] and response time. Time is usually measured directly in milliseconds. Effectiveness measures, on the other hand, can vary. Therefore, this section focuses on describing the most commonly used similarity measures in Ad Hoc IR. These measures are defined based on the following parameters:

- True Positive (vp): Number of relevant documents that are retrieved.

[4]Measure of how much and how the system retrieves relevant documents

- True Negative (vn): Number of non-relevant documents that are not retrieved.
- False Positive (fp): Number of non-relevant documents that are retrieved.
- False Negative (fn): Number of relevant documents that are not retrieved.

2.7.1 Precision and Recall

The first two relevance measures proposed are Precision (P) and Recall (R). Precision evaluates the number of relevant documents retrieved by the search engine relative to the total number of retrieved documents and is computed by Eq. 2-18:

$$P = \frac{vp}{vp + fp} \tag{2-18}$$

Recall calculates how many relevant documents are retrieved relative to the total number of relevant documents and is calculated by Eq. 2-19:

$$R = \frac{vp}{vp + fn} \tag{2-19}$$

Some variations of these two measures appear in some articles, aiming to evaluate the number of relevant documents in the top k positions of the ranking generated by the system. These measures are P@k (Precision at position k) and R@k (Recall at position k). They correspond to the precision obtained in the top k elements of the ranking and the recall obtained in the top k elements of the ranking.

2.7.2 Average Precision

Average Precision (AP) corresponds to the average of the precision values at each position in the list of retrieved documents. If the system retrieves m documents, then the average precision is given by Eq. 2-20:

$$AP = \frac{1}{m} \sum_{k=1}^{m} P@k \tag{2-20}$$

2.7.3 Mean Average Precision

Mean Average Precision (MAP) is the mean of the average precision of the queries in a set of queries Q . The calculation of the measure is shown in Eq. 2-21:

$$MAP = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} p_{kj} \tag{2-21}$$

where $|Q|$ is the size of the query set.

2.7.4 Reciprocal Rank

The Reciprocal Rank (RR) is a statistical measure used for evaluating the list of the retrieved documents relevant to a query set and ordered by correctness. The RR is computed as the inverse of the position rel_i in the ranking where the first relevant document for the query occurs, shown in Eq. 2-22:

$$RR = \frac{1}{rel_i} \tag{2-22}$$

Reciprocal Rank shows how well the IR system can bring relevant documents closer to the top of the ranking.

2.7.5 Mean Reciprocal Rank

Mean Reciprocal Rank (MRR) is a measure that corresponds to the average of RR for a set of queries Q. MRR takes into consideration the first relevant result to a query and ignores all other relevant documents. MRR is calculated using Eq. 2-23:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} RR_i \tag{2-23}$$

where $|Q|$ is the size of the query set, Q denotes the total number of queries and RR_i is the inverse of the rank of the first relevant document to a query q.

2.7.6 Discounted Cumulative Gain

Discounted Cumulative Gain (DCG@k) is a measure introduced to evaluate the discount in gain (a penalty) when the system retrieves a relevant document far from the top positions of the ranking, considering the top k documents in the ranking. DCG@k is used to evaluate rankings in web search engines because, for these IR systems, it is important that they bring the most relevant documents to the top positions of the ranking.

DCG@k is based on the logarithmic penalty of any highly relevant document that appears in a position far from the top of the ranking. It can be used when the notion of relevance is graded, can assume values other than 0 or 1. DCG@k is calculated by accumulating the graded relevance at each position of the ranking starting from the first position, as shown in Eq. 2-24:

$$DCGp@k = \sum_{i=1}^{k} \frac{2^{rel_i - 1}}{log_2(i=1)} \tag{2-24}$$

where rel_i is the graded relevance of the document at position i of the ranking.

2.7.7 Normalized Discounted Cumulative Gain

DCG@k does not generate a normalized value. To obtain a normalized value, between 0 and 1, the Normalized Discounted Cumulative Gain (nDCG@k) is often used, which is based on an ideal DCG@k (i.e., IDCG@k).

IDCG@k generates the maximum possible DCG value at each position i, $1 \leq k$, that is, the ideal ordering of relevant documents that appear in the top k positions is obtained to maximize DCG@k. The result of this operation corresponds to IDCG@k - ideal DCG. Then for a given query, nDCG@k is calculated using Eq. 2-25:

$$nDCG@k = \frac{DCG@k}{IDCG@k} \tag{2-25}$$

nDCG@k is computed for a query. Typically, the average of nDCG@k for a set of queries Q is reported.

2.8 Related Work

2.8.1 Poly-encoder

Humeau et al. [27] introduced a new architecture called Poly-encoder, aiming to improve the effectiveness of bi-encoder models and reduce the latency of cross-encoder models. The authors proposed an additional learned attention mechanism that represents more global attributes (or features) from which self-attention can be performed. This proposal was tested on dialogue response selection and article retrieval tasks, resulting in performance gains over bi-encoder baselines and significant speed improvements over cross-encoders.

The datasets used are ConvAI2, DSTC7, Ubuntu V2, and Wikipedia Article Search. ConvAI2 is based on Persona-Chat and involves dialogues between pairs of speakers with specific fictional personas; that is, each speaker is given a persona with a few words and must imitate that persona in a dialogue. Based on this dialogue history, the model must choose the correct response from 20 options, of which only one is correct. DSTC7 (Track 1) consists of conversations extracted from Ubuntu chat logs, where one partner receives technical support on various system-related issues. Ubuntu V2 is a version of DSTC7 with more data. The Wikipedia Article Search uses a sentence extracted from a Wikipedia article as a query to find the original article.

For better understanding, we use the terms "query" and "documents" instead of the terms "context" and "label," respectively, used by the authors of Poly-encoder. In ConvAI2, the documents are the options that contain few tokens compared to the query (the dialogue). In DSTC7 and Ubuntu V2, the documents are the related issues.

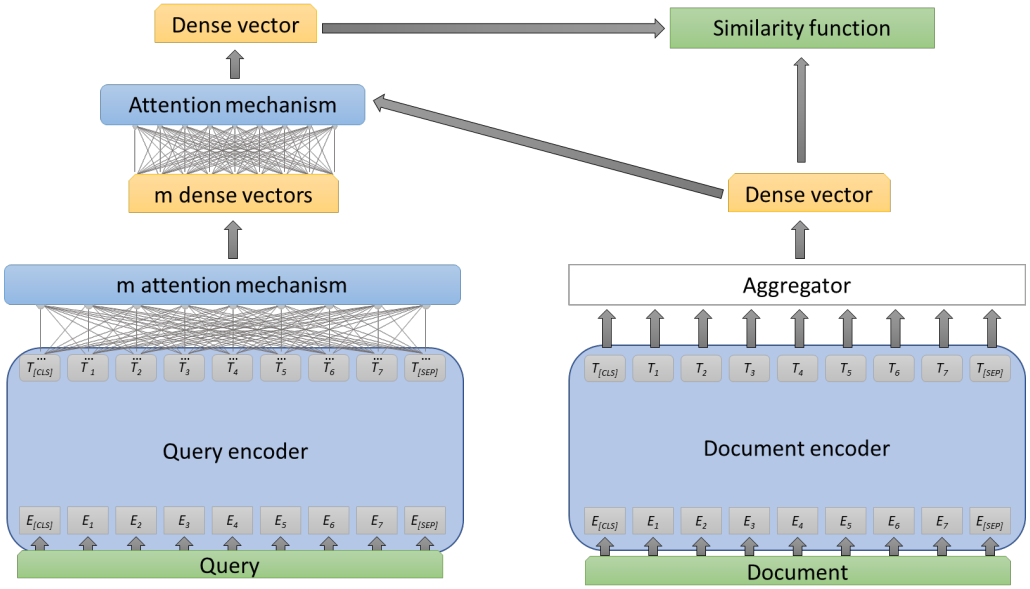


Figure 2.8: *General architecture of the Poly-encoder. [27]*

In Wikipedia Article Search, the documents are the articles, and an arbitrary segment of these is the query.

Considering these datasets, the Poly-encoder generates a single dense vector for the document. Humeau et al. [27] conducted extensive experiments with three aggregation strategies: selecting the contextualized token T[CLS] from BERT output, averaging all BERT output token vectors, or averaging the first $m < N$ BERT output token vectors, where N is the total number of tokens and m is a parameter referring to the desired number of tokens.

The experimental results indicate that the output token T[CLS] performs best in terms of effectiveness. Therefore, the Poly-encoder uses two independent encoders for the query and document. The general architecture of the model is shown in Figure 2.8. According to the characteristics of used datasets, the dialogue is a query that is naturally longer than the document in terms of text length; for this reason, it is represented by m global dense vectors $(v_1, \ldots, v_m)$. Meanwhile, the document is represented by the BERT output vector T[CLS].

To obtain these m global dense vectors v_i representing the input, m query codes $(c_1, \ldots, c_m)$ are learned during fine-tuning, initialized randomly, where c_i extracts the vector v_i from the query vectors, considering the importance of all outputs from the previous layer. The attention weights $w_j^{c_i}$ are a probability distribution over the outputs of the previous layer, determining the relative importance of each output token vectors[5]

[5]Note that in latter pruning method by Lassance et al. [32] is used the last BERT layer to determine the relative importance of each token in input sequence to retain top-k document tokens, explained in more details in subsection 2.8.8.

for the global vector. Thus, v_i is obtained using Eq. 2-26:

$$v_i = \sum_j w_j^{c_i} \cdot h_j \qquad \text{where} \quad (w_1^{c_i}, \ldots, w_N^{c_i}) = \text{softmax}(c_i \cdot h_1, \ldots, c_i \cdot h_N) \tag{2-26}$$

where h_i are the BERT dense vectors.

Given the m global dense vectors of the query $(v_1, \ldots, v_m)$, attention is applied to them using the dense vector T[CLS] of the document. This process generates a dense vector v that represents the query, as shown in Eq. 2-27:

$$v = \sum_i w_i \cdot v_i \qquad \text{where} \quad (w_1^{c_i}, \ldots, w_N^{c_i}) = \text{softmax}(T_{[CLS]_i} \cdot v_1, \ldots, T_{[CLS]_i} \cdot v_m) \tag{2-27}$$

The relevance score is calculated between the dense vector v of the query and the dense vector T[CLS] of the document, using the dot product as the similarity function. Humeau et al. [27] explored other alternatives for selecting the m dense vectors from the set of BERT output vectors:

- Consider the first m dense vectors of the output, apply Eq. 2-27, and compute the similarity.
- Consider the last m dense vectors of the output, apply Eq. 2-27, and compute the similarity.
- Consider the last m dense vectors of the output concatenated with the dense vector T[CLS], apply Eq. 2-27, and compute the similarity.

The results indicate that the Poly-encoder, with m global dense vectors $(v_1, \ldots, v_m)$, performs best in terms of effectiveness. However, considering the first m dense vectors achieves competitive results. Additionally, in many IR tasks, queries are shorter than documents.

2.8.2 ME-BERT

The proposal by Humeau et al. [27] demonstrated improvements in effectiveness over bi-encoder baseline models and significant efficiency gains over cross-encoder models for dialogue response selection and article retrieval tasks. However, the specific characteristics of the IR task and its respective dataset were completely ignored, resulting in assumptions contrary to the facts (e.g., queries longer than documents). In the dataset description, the authors of Poly-encoder highlight that, in the IR task, a sentence extracted from a Wikipedia article is used as a query to find the original article. Naturally, the query is shorter in terms of text length compared to the document.

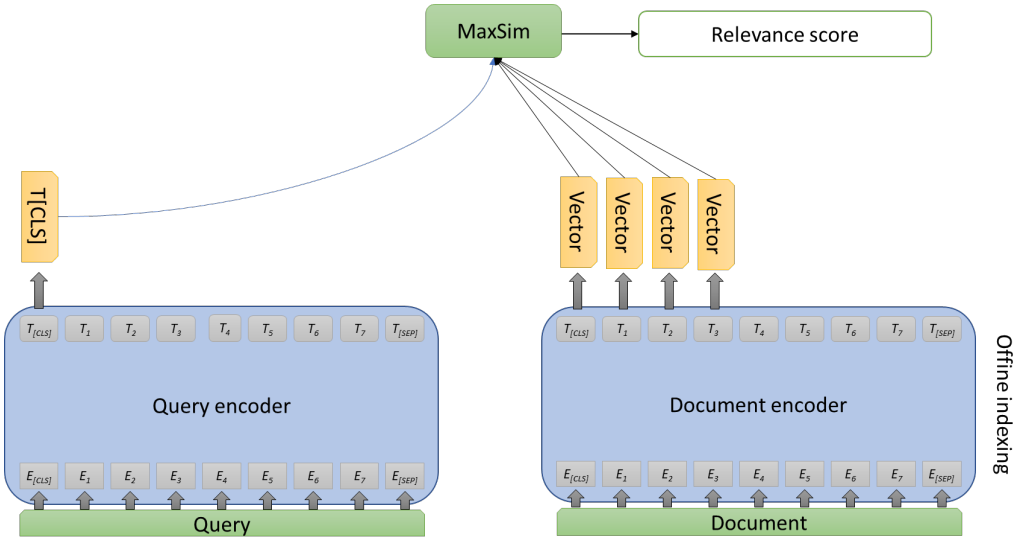


Figure 2.9: *General architecture of ME-BERT [44]. The query is represented by the dense vector T[CLS]. The document is represented by m dense vectors, where the parameter* $m = 4$.

Due to the use of attention mechanisms in Poly-encoder, it is not possible to decompose the relevance score into a maximum over dot products. As a result, fast nearest neighbor search cannot be applied, making efficient document retrieval across the entire collection challenging.

Luan et al. [44] proposed ME-BERT (Multi-Vector Encoding from BERT), with the goal of investigate the capacity of the bi-encoder architecture compared to sparse bag-of-words (BoW) models and attention-based neural networks. The investigation is conducted both theoretically and empirically. From the theoretical analysis, the authors concluded that the size of document vector representations may, in general, need to be larger for long documents. We refer the reader to Luan et al. [44] for more detailed theoretical analyses.

These theoretical results motivated Luan et al. [44] to propose a new architecture that represents each document as a set of the first m output vectors from BERT, while maintaining only one vector (T[CLS]) for the query. Relevance scores are calculated as the maximum dot product between the set of m vectors and the query vector, as shown in Figure 2.9.

Thus, the relevance of a document d from a collection $\mathcal{C}$ w.r.t a given query q is formulated as follows:

$$\text{rel}(q,d) = \max_{j \in m} T_{[CLS]_q} \cdot v_{j_d} \tag{2-28}$$

where v_{j_d} is the j-th dense vector of document d and m corresponds to the first m dense vectors of document d. The computation of similarity between a query and the

documents in the collection involves the dot product operation. This operation can be efficiently implemented using approximate nearest neighbor search techniques [44]. In this approach, m dense vectors are added to the index for each document in the collection.

The process of determining the relevance of a document d w.r.t a query q can be performed in two steps:

First Step: ME-BERT retrieves the k most similar documents to the query q using nearest neighbor search in the index.

Second Step: The model refines the relevance by applying Eq. 2-28 to the documents retrieved in the first step.

This two-step approach allows for efficient computation of similarity (parallel on GPU) between the query and the documents, avoiding the need to compute the exact relevance between the query and all documents in the collection. By restricting the computation to the k nearest documents, the model can return the ranking more quickly and scalably, especially for large collections.

The ME-BERT was trained using a combination of negatives (non-relevant documents) retrieved by BM25 and in-batch negatives, employing cross-entropy loss as the objective function. Moreover, an additional round of hard negative mining was applied in some experimental scenarios.

Experimental results demonstrate that the ME-BERT outperforms its single-vector version, DE-BERT (with $m = 1$, where the [CLS] token is used for the document vector), as well as the DPR [29]. However, ME-BERT does not achieve as high performance as the ANCE [82], which came later and also uses dense vectors.

Luan et al. [44] also combine the results of dense retrieval (based on vector representations) with sparse retrieval from BM25, using a linear combination of relevance scores to build an effective dense-sparse hybrid model. This hybrid approach is similar to other recent studies in the field of information retrieval.

2.8.3 ColBERT

The approaches Poly-encoder [27] and ME-BERT [44] demonstrated significant gains in terms of effectiveness, raising a question in the IR community [39]: if generating multiple vectors for query and document tokens is promising, why not take it to the extreme and generate a dense vector for each token?

Motivated by the idea of generating multiple vectors for a query or document, the ColBERT (Contextualized Late Interaction over BERT) [30] was proposed. This model generates dense vectors for each token in the query and document (including the [CLS] tokens of both) and uses a late interaction mechanism, combining representation-based and interaction-based approaches.

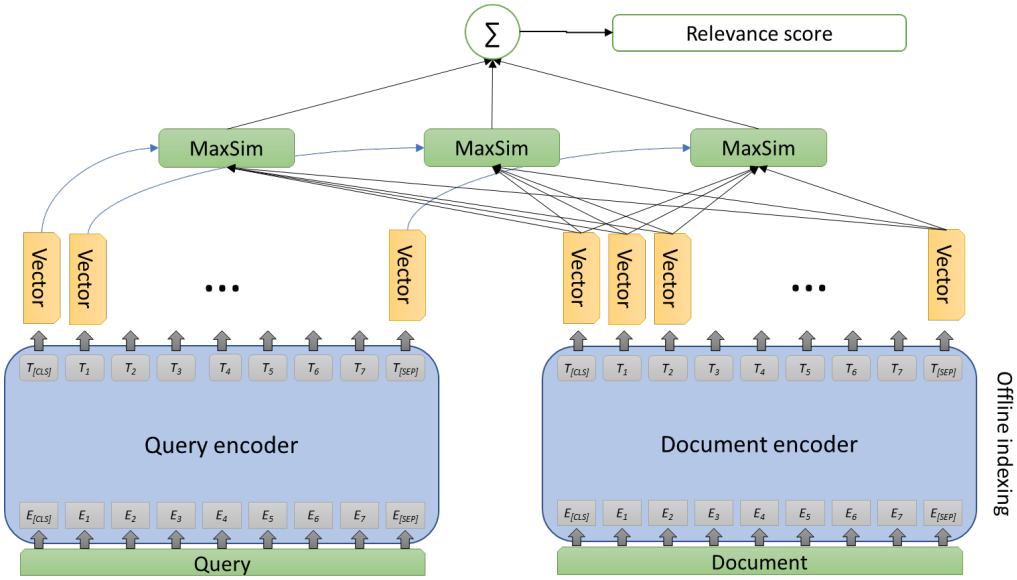


Figure 2.10: *General architecture of ColBERT. [30]*

The interaction component of ColBERT is implemented through a function called summation of maximum similarity, as shown in Figure 2.10. This function computes the cosine similarity between each query token vector and all document token vectors, selecting the maximum similarity for each query vector. The document relevance score is given by the sum of these maximum similarities. The authors call this operation late interaction, as it allows rich interactions between query tokens and document tokens in a way compatible with existing nearest neighbor search techniques.

Given a sequence of tokens $s_t = [t_1, \ldots, t_n]$, ColBERT uses a transformation $\eta : [t_1, \ldots, t_n] \rightarrow \mathbb{R}^{n \times D}$, where D is the dimension of the dense vectors for each token. That is, η generates a matrix of dimension $n \times D$.

Khattab and Zaharia [30] use the same BERT to encode queries and documents in the collection. To distinguish query from document token sequences, they place a [Q] token between [CLS] and the first query token, and a [D] token between [CLS] and the first document token. Both inputs end with the [SEP] token.

Given a query q, if q has fewer tokens than a predefined number of tokens N_q, the missing tokens are filled with the [MASK] token. This allows BERT to produce dense vectors in the positions corresponding to these masks, enabling query expansion with new tokens or adjusting the importance of document tokens w.r.t the query.

To control the dimension D, a network layer, CNN, is added to the last layer of BERT. Typically, D is a fixed number, smaller than the dimension of the contextualized token vectors generated by BERT. The contextualized dense vectors of the query and documents are normalized using the L2 norm. This allows using only the dot product to compute the cosine similarity between the vectors.

The procedure used in the query encoder is similar to the process adopted in

the document encoder, except that the [MASK] token is not used in the document. After passing a sequence of tokens s_t through BERT and the subsequent linear layer, followed by normalization, the document encoder filters the dense vectors corresponding to punctuation symbols, determined through a predefined list. This filtering aims to reduce the number of dense vectors per document, as the authors of ColBERT hypothesize that dense vectors (even contextualized) of punctuation are unnecessary for effectiveness.

Formally, the dense vectors of the query q and document d are presented in Eq. 2-29. Given $q = [q_1, q_2, \ldots, q_{|q|}]$ and $d = [d_1, d_2, \ldots, d_{|d|}]$, where q_i and d_i are tokens forming the query q and document d, respectively, ColBERT computes the sets of dense vectors (bag of embeddings) (matrices) E_q and E_d:

$$\begin{aligned} E_q &= \text{L2}(\text{CNN}(\text{BERT}([\text{CLS}][\text{Q}]q_1 \ldots q_{|q|}, [\text{MASK}]_{|q+1|} \ldots [\text{MASK}]_{N_q}, [\text{SEP}]))) \\ E_d &= \text{Filter}(\text{L2}(\text{CNN}(\text{BERT}([\text{CLS}][\text{D}]d_1 \ldots d_{|d|}, [\text{SEP}])))) \end{aligned} \tag{2-29}$$

Once the sets of dense vectors E_q and E_d are obtained, the similarity between them is calculated using the summation of maximum similarities 2-30:

$$\text{sim}(q,d) = \sum_{i=1}^{|q|} \max_{d_j \in d} (E_{q_i} \cdot E_{d_j}^{\text{T}}) \tag{2-30}$$

During training, for each query q, ColBERT uses a positive (relevant) document d^+ and a negative (non-relevant) document d^-. The similarities $\text{sim}(q,d^+)$ and $\text{sim}(q,d^-)$ are calculated, and the following loss function is used:

$$\mathcal{L}(q,d^+,d^-) = -\log \frac{e^{\text{sim}(q,d^+)}}{e^{\text{sim}(q,d^+)} + e^{\text{sim}(q,d^-)}} \tag{2-31}$$

At inference, ColBERT can be used both as a re-ranker and as a first-stage IR model. In end-to-end retrieval, ColBERT requires an immense number of comparisons due to its late interaction mechanism. Therefore, Khattab and Zaharia [30] use a two-step procedure to retrieve the top k documents from the entire collection, similar to ME-BERT [44]. Both steps depend on the ColBERT score: the first is an approximate search phase for filtering, while the second is a refinement phase. The authors relax the maximum similarity in the first step, using Faiss [28], an efficient library for vector comparison, but it does not guarantee that the exact maximum similarity vector is obtained.

For the retrieval of top-k documents given a query q with N_q tokens, the process is described as follows: given a token vector E_{q_i} from this set of N_q tokens, the Faiss index is searched for $K' = \frac{K}{2}$ vectors, where K is the total number of vectors in index. This retrieves the top K' dense vectors most similar to the query token vector E_{q_i}. Based on these retrieved vectors, the source documents are located, producing $N_q \times K'$ source

documents, of which only $k \leq N_q \times k'$ are unique. These k documents likely contain one or more dense vectors that are highly similar to the query vectors. In the second step, ColBERT refines this set by exhaustively re-ranking only the k documents.

Although the ColBERT shows 3.8% lower effectiveness compared to re-ranking with the monoBERT$_{\text{LARGE}}$, it is 70 times faster in execution. Additionally, ColBERT shows an improvement in recall for the top 1,000 documents compared to BM25. This indicates that using ColBERT allows retrieving relevant documents that BM25 cannot identify.

Khattab and Zaharia [30] also analyzed the importance of different components of the model relative to the MRR@10 metric:

- Using the dense vectors of T[CLS] for the query and T[CLS] for the document, instead of the vectors for each token, results in a drop of approximately 20%, even though T[CLS] vectors are passed through an output layer that increases the vector dimension.
- Using the average of similarities instead of the maximum similarities results in a drop of approximately 20%.
- Not using the [MASK] token in the empty positions of the query results in a drop of 6%.

The space occupied by the vectors varies according to their dimensions and the number of bytes occupied by each float in a dimension. Reducing the number of dimensions and the number of bytes per dimension does not cause a substantial loss of effectiveness (MRR@10). Between the most effective and least effective configurations, there is a 6% loss in effectiveness but allows a 95% reduction in space occupied.

The results show that, in terms of latency, ColBERT is an intermediate solution between monoBERT and pre-BERT neural models. This gain in time is achieved with a small loss of effectiveness compared to monoBERT used in re-ranking. However, monoBERT is only a baseline; there are more effective models for re-ranking. Still, ColBERT is many times slower than pre-BERT neural models and doc2query (BM25 with document expansion applied).

2.8.4 ColBERTv2

ColBERTv2 [74] is a more recent and improved version of the ColBERT [30], with changes in four main aspects: a) changes in training; b) changes in representation; c) changes in indexing and storage of dense token vectors; and d) changes in retrieval. These modifications make ColBERTv2 significantly more effective (in MRR@10) and more efficient, both in terms of inference time and the space occupied by the index and dense vectors, compared to ColBERT [30].

Training

More recent dense and sparse learning methods have adopted three approaches to improve the training of bi-encoders:

- Negative Mining: Involves finding better negative examples.
- Pre-training: Involves training a model on a dataset different from the one where the model's inference will be applied, i.e., training before fine-tuning for a specific task.
- Distillation: Involves using a teacher model to train a student model.

To train a bi-encoder model, positive and negative documents are typically required for each query in the training set. The triple-based loss function (q, d^+, d^-) used by ColBERT was identified by later work as having several weaknesses. These weaknesses stem from the fact that ColBERT uses one positive document per query, annotated by a human, and a negative document obtained from the top of the BM25 retrieval list. These negative examples are not informative and tend to bias the model to mimic BM25's sparse retrieval [82].

To address these weaknesses, ColBERTv2 uses negative mining and distillation to improve the effectiveness of the trained model. Thus, ColBERTv2 uses ColBERT trained with triples to identify hard negatives (better negative examples).

For each training query, the top-k documents are retrieved using ColBERT. Then, query-document pairs are formed and passed through the layers of a cross-encoder to re-rank the documents. For each query, a w-tuple is formed, similar to a triple, where, in addition to the query, a positive document (from the top of the ranking) and one or more negative documents from the bottom of the ranking are included. A collection is formed by all queries, consisting of $w = 64$ documents.

Given the collection $w = 64$ and the scores produced by ColBERT and the cross-encoder, the KL-Divergence loss function is used to distill the cross-encoder scores into the ColBERT architecture. Santhanam et al. [74] also use in-batch negatives per GPU and apply the cross-entropy loss function, using the positive example of the query against all documents corresponding to other queries in the batch, which are used as d^-.

Representation, Indexing, and Storage of Dense Vectors

ColBERTv2 uses a technique called residual compression to reduce the amount of memory required to store the vectors for late interaction. Santhanam et al. [74] hypothesizes that document token vectors cluster in regions that capture the semantics of the tokens. Specifically, vectors corresponding to each sense of a word cluster, differing in minimal contextual variations. ColBERTv2 obtains a set of centroids C w.r.t the token

vectors generated for the documents. A vector v of a token is encoded with the index t of the nearest centroid C_t and a bit-quantized vector $\tilde{r}$ that approximates the residual $r = v - C_t$.

ColBERTv2 compresses each dimension of the residual vector r into one or two bits b. In principle, encoding an n-dimensional vector requires $d \log |C| + bn$ bits per vector. A vector v of a token is encoded with the index t of the nearest centroid C_t and a bit-quantized vector $\tilde{r}$ that approximates the residual $r = v - C_t$. In practice, with $n = 128$, 4 bytes are used to represent up to 2^{32} centroids and 16 or 32 bytes (for $b = 1$ or $b = 2$) to encode the residual. Thus, a vector occupies 20 or 36 bytes, in contrast to the 256 bytes occupied by each vector in ColBERT.

Indexing in ColBERTv2 is divided into three stages:

- Centroid Selection: ColBERTv2 defines a number of centroids $|C|$ proportional to $\sqrt{E}$, where E is the total number of dense token representations in the documents[6]. Centroids are generated after processing approximately $\sqrt{E}$ documents using the k-means method.
- Document Compression: In this stage, documents are processed by ColBERTv2 to generate the corresponding vectors and saved in the index.
- Index Inversion: The IDs of dense vectors associated with each centroid are grouped and saved in the index to support more efficient search.

Retrieval

ColBERTv2 generates rankings in two stages, similar to ColBERT. In the first stage (approximate search), the k documents are selected considering only the dense vectors of tokens grouped in some centroids. That is, the score for this phase is generated through vectors grouped in some centroids. In the second stage (refinement), since the vectors of a document may be distributed across multiple centroids, including centroids farther from the query, they are regrouped, and the similarity is calculated using all vectors of the document.

Formally, given a query q, ColBERTv2 starts with an approximate search to generate candidates. For each query token q_i, the k nearest centroids are identified. Using inverted lists, ColBERTv2 retrieves the dense vectors of documents associated with each centroid, decompresses them, and calculates the similarity with each query vector. The scores are grouped by document ID for each query token vector, and the maximum similarities associated with the same document ID are obtained, as in ColBERT [74].

[6] The next power of 2 greater than or equal to $16 \cdot \sqrt{E}$

Once this similarity is calculated, a ranked list is generated where k documents are selected for refinement. All dense vectors of each document are loaded to compute the similarity of all query tokens with all document tokens, using the same equation as ColBERT, Eq. 2-30. Finally, the documents are ordered by score, and the top k are presented to the user.

2.8.5 PLAID

PLAID [73] extends ColBERTv2 [74] with the PLAID (Performance-optimized Late Interaction Driver), designed to accelerate query processing for models using the late interaction architecture of ColBERT [30], which includes systems like ColBERTv2. PLAID enhances retrieval efficiency by leveraging ColBERTv2's centroids through a multi-stage filtering strategy that progressively narrows down candidate documents using centroid-based interactions as well as centroid pruning, a mechanism for sparsifying the bag of centroids.

The approach begins by representing documents as bags of centroid IDs, delaying the loading of full residuals. By exclusively utilizing centroid-only multi-vector retrieval, which offers high recall without using the vector residuals, PLAID efficiently identifies high-scoring documents while discarding weaker candidates. This strategy allows for skipping low-scoring documents without having to look up or decompress their residuals. Moreover, since not all centroids contribute equally to relevance determination for a given query, PLAID prioritizes computations over highly weighted centroids and discards the rest. Final relevance scores are computed over a significantly reduced candidate set, resulting in substantial improvements in query response times for large-scale retrieval.

Latency Analysis of ColBERTv2

The total latency of ColBERTv2 is divided into five components: query encoding, candidate generation, index lookups (i.e., to gather the compressed vector representations for candidate documents), residual decompression, and final MaxSim computations [73].

Santhanam et al.[73] observed that in ColBERTv2, gathering vectors from the index is expensive because it consumes significant memory band width: each vector in the model is encoded with a 4-bit centroid ID and 32-byte residuals, each document contains dozens of vectors, and there can be up to 2^6 candidate documents. Additionally, index lookup in ColBERTv2 also constructs padded tensors on the fly to deal with the variable length of passages. Residual decompression involves several non-trivial operations, such as bit unpacking and large-scale summations, which can be computationally expensive

when ColBERTv2 generates a large initial candidate set (∼10–40k documents). While it is possible to use a smaller candidate set, doing so reduces recall. Based on this finding, Santhanam et al.[73] study how to optimize late-interaction search latency at a large scale, taking all steps of retrieval into account.

Centroids Identify Strong Candidates

Santhanam et al.[73] demonstrate through experiments that centroids can accelerate retrieval without compromising quality by acting as proxies for document vectors. PLAID first retrieves candidates by matching the closest ColBERTv2 centroids with the query token vectors. Since centroids are drawn from a fixed set, the distance between query vectors and all centroids can be computed once during retrieval and reused across all document representations (bag of centroids). This allows low-scoring documents to be skipped without the need to fetch or decompress their residuals, introducing only a minor overhead in candidate generation while yielding substantial efficiency gains in later stages.

Centroid Importance per Query

PLAID [73] showed that not all centroids play a significant role in every query, which means that the importance of a centroid is directly related to its similarity to the query tokens. In many cases, only a fraction of the centroids contributes meaningfully to relevance estimation. Therefore, highly weighted centroids can be prioritized while discarding the rest, resulting in sparse document representations.

PLAID pipeline

The PLAID [73] pipeline consists of four stages: candidate generation, centroid interaction with centroid pruning, centroid interaction without centroid pruning, and scoring.

Candidate Generation

Candidate generation produces an initial set of candidate documents by computing relevance scores for each centroid w.r.t the query vectors. Formally, given the query vector matrix Q and the list of centroid vectors C from the index, PLAID calculates token-level query-centroid relevance scores $S_{c,q}$ as a matrix multiplication:

$$S_{c,q} = C \cdot Q^T \tag{2-32}$$

Once the relevance scores are calculated, the documents closest to the top-k centroids are identified to form the initial set of candidate documents. However, a document is considered close to a centroid if at least one vector of the document is in the centroid.

Centroid Interaction

Centroid interaction cheaply computes approximate document relevance by replacing each token vector with its nearest centroid in the standard MaxSim formulation. Because the centroids come from a fixed set, this stage reuses the relevance scores for each centroid w.r.t the query tokens stored in $S_{c,q}$ from Eq. 2-32 across all bag-of-centroids document representations. Formally, suppose I is the list of centroid indices mapped to each of the tokens in the candidate set. Moreover, let $S_{c,q}[i]$ be the i-th row of $S_{c,q}$. Then, PLAID constructs approximate centroid-based scores $\tilde{D}$ as:

$$\tilde{D} = \begin{bmatrix} S_{c,q}[I_1] \\ S_{c,q}[I_2] \\ \vdots \\ S_{c,q}[I_{|\tilde{D}|}] \end{bmatrix} \tag{2-33}$$

A document will consist of a set of centroids, each of which is closest to its respective token vector. Since the relevance scores for each centroid w.r.t the query tokens are stored in $S_{c,q}$ from Eq. 2-32, MaxSim is applied to each query vector with all centroids of the document without the need to re-compute the dot product. That is, to rank the candidate documents using $\tilde{D}$, PLAID calculates the MaxSim scores $S_{\tilde{D}}$ as:

$$S_{\tilde{D}} = \sum_{i=1}^{|Q|} \max_{j=1}^{|\tilde{D}|} \tilde{D}_{i,j} \tag{2-34}$$

The top-k most relevant documents extracted from $S_{\tilde{D}}$ serve as the filtered candidate document set. Thus, centroid interaction serves as an additional filtering stage, allowing PLAID to skip the expensive process of vector reconstruction for a large fraction of candidate documents. Intuitively, centroid interaction allows PLAID to emulate traditional bag-of-words retrieval, where centroid relevance scores take on the role of term relevance scores used in systems like BM25. However, due to its vector representation (particularly of the query), PLAID calculates centroid relevance scores at query time, in contrast to more traditional term relevance scores that are pre-computed.

Centroid Pruning

PLAID first prunes centroids with low-magnitude scores w.r.t a given threshold t_{cs} before constructing $\tilde{D}$. That is, $\tilde{D}$ will consist only of tokens whose corresponding centroid (suppose centroid i) satisfies the following condition:

$$\max_{j=1}^{|\tilde{D}|} S_{c,q_{i,j}} \geq t_{cs} \tag{2-35}$$

PLAID introduced three main hyperparameters:

- nprobe defines the number of nearest centroids in candidate generation.
- t_{cs} acts as a threshold for centroid selection, removing those whose maximum similarity with all query tokens is below this value.
- ndocs specifies the number of candidate documents selected during centroid interaction, both with and without pruning. Experimentally, the value $ndocs/4$ proved most effective for the third stage of the PLAID pipeline.

Scoring

PLAID loads and reconstructs all residuals for each document in the candidate document set to compute the similarity of each query tokens with all document tokens, using the same equation as ColBERT, Eq. 2-30. Finally, the documents are ordered by score, and the top k are presented to the user.

2.8.6 COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List

COIL (Contextualized Inverted List) [19] is a retrieval method that integrates dense token representations into traditional inverted index structures. Inspired by ColBERT, document-side token embeddings are pre-computed offline. During query time, instead of computing similarity between each query token and all document token embeddings, COIL restricts the comparison to only those document token vectors that are indexed under the same token in a standard inverted list. Each posting entry consists of a document identifier and the dense embedding of the token in that document. Thus, each inverted list effectively becomes a list of dense vectors.

A limitation of this approach is its vulnerability to the *vocabulary mismatch* problem, as the token-level similarity is only computed for exact lexical matches—i.e., the system accesses only the embeddings corresponding to query terms. To mitigate this issue, COIL introduces a complementary mechanism that incorporates the contextual embeddings of entire queries and documents. Specifically, the method utilizes the output

embeddings of the [CLS] token—denoted as q_{cls} and d_{cls}—to enrich document-query similarity computation.

The similarity between a query q and a document d is initially defined using a token-level matching score:

$$S_{\text{tok}}(q,d) = \sum_{q_i \in q} \max_{d_j = q_i} (q_i^\top d_j) \tag{2-36}$$

This formulation relies solely on exact token matches and does not address semantic similarity across different lexical forms. To compensate, Gao et al. [19] propose an augmented similarity function $S_{\text{full}}(q,d)$, which incorporates a dense-level similarity based on contextual embeddings:

$$S_{\text{full}}(q,d) = S_{\text{tok}}(q,d) + q_{\text{cls}}^\top d_{\text{cls}} \tag{2-37}$$

The additional dot product introduces minimal overhead but significantly improves robustness against vocabulary mismatch. Both q_{cls} and d_{cls} embeddings are projected to a 768-dimensional space prior to use.

From an implementation perspective, the inverted list for each token t is stored as a matrix $\mathbb{R}^{n \times |I_t|}$, where n is the embedding dimension and $|I_t|$ is the number of occurrences of t across documents. This structure enables efficient computation of inner products using optimized BLAS operations on both CPUs and GPUs. The contextual document vectors d_{cls} are also organized into matrices for similar computational benefits. While exact search is supported, the authors suggest that additional efficiency can be gained by approximating the inverted index with approximate nearest neighbor (ANN) techniques.

Training is conducted via a negative log-likelihood objective. Given a query q, a relevant document d^+, and a set of l non-relevant documents $\{d_1^-, \ldots, d_l^-\}$, the loss function is defined as:

$$\mathcal{L} = -\log \left(\frac{\exp(S(q,d^+))}{\exp(S(q,d^+)) + \sum_{i=1}^{l} \exp(S(q,d_i^-))} \right) \tag{2-38}$$

Here, S may correspond to either S_{tok} or S_{full}, depending on the training variant. Negative samples are drawn from the top 1000 BM25-ranked documents that are not labeled as relevant, with each query being paired with one positive and seven negative examples (i.e., $l = 7$).

2.8.7 What are you token about?

Single-vector models have proven effective in generating rankings. However, when applied without fine-tuning on a specific collection, a significant loss of effective-

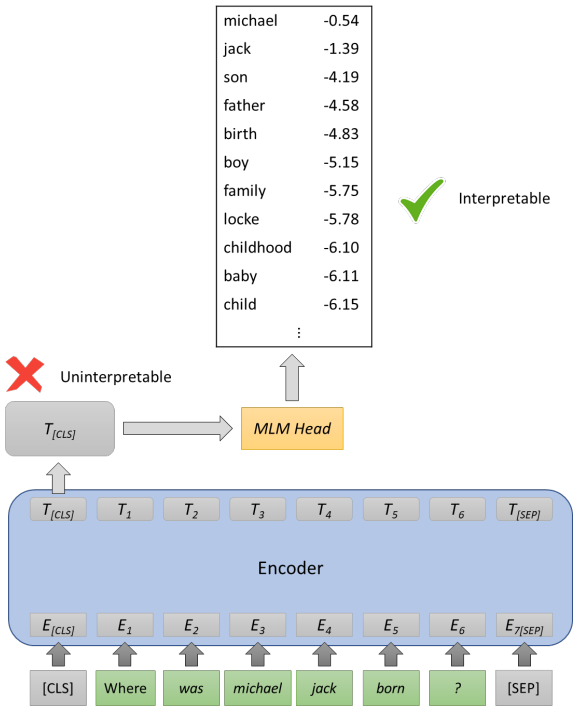


Figure 2.11: *An example of projection. The query "Where was Michael Jack born?" is passed through the DPR query encoder (BERT), and the contextual vector $T_{[CLS]}$ is projected into the vocabulary space using BERT's MLM head. The result is a distribution over the vocabulary.*

ness is observed. The reasons for this loss are not yet well understood, as the information captured by the single-vector representation has not been thoroughly investigated.

The authors Ram et al. [67] present an approach to interpret the dense vector produced by BERT or associate the dense vector with the tokens in BERT's vocabulary. The approach obtains a distribution of vocabulary tokens induced by projecting the dense vectors of both documents and queries into the vocabulary space. Such projections allow for a better understanding of dense vectors and their deficiencies, highlighting which vocabulary tokens are most related to a given dense vector.

The projections are human-interpretable. In these projections, it is possible to observe a significant overlap between the top k tokens with the highest weights in both document and query projections (as in traditional sparse models). This observation suggests that lexical overlap between query and document plays an important role in retrieval. The top k tokens in the projections of relevant documents have a high probability of occurring in the corresponding query and may represent predictions of relevant queries for the document. Therefore, the projections can be used for query expansion.

These findings are surprising because the models are trained contrastively and

do not make any predictions about the vocabulary or even use the MLM head. Thus, the vocabulary token distributions are entirely related to the MLM task trained by the model during pre-training. The paper [67] partially explains why dense retrievers fail in some entity-centric queries (a problem already reported in the literature).

The MLM head is a function that takes a vector $h \in \mathbb{R}^d$ and returns a distribution P over the model's vocabulary $\mathcal{V}$, defined as follows:

$$\text{MLM-Head}(h)[i] = \frac{e^{(v_i^{\mathrm{T}} g(h))}}{\sum_{j \in \mathcal{V}} e^{(v_j^{\mathrm{T}} g(h))}} \tag{2-39}$$

where $g : \mathbb{R}^d \to \mathbb{R}^d$ is a non-linear function (a fully connected layer followed by a normalization layer). $v_i \in \mathbb{R}^d$ is the static vector of the i-th token in $\mathcal{V}$.

Together, the values MLM-Head$(h)[i]$, for all $i \in \mathcal{V}$, form a probability distribution over $\mathcal{V}$ w.r.t h. Let Enc_d be the encoder used to generate the document vector e_d, and let Enc_q be the encoder used to generate the query vector e_q. The two projections into the vocabulary are obtained as shown in Eq. 2-40:

$$\begin{aligned} Q &= \text{MLM-Head}(e_q) \\ D &= \text{MLM-Head}(e_d) \end{aligned} \tag{2-40}$$

Initially, it is not guaranteed that Q and D have meaning, as the encoder models changed after pre-training, while the MLM head was not modified during fine-tuning. Moreover, the MLM head was not trained to decode [CLS] or mean aggregation. Despite this, Q and D are very intuitive and facilitate the understanding of single-vector models.

In the study, is observed that 71% of the top k tokens in Q appear in both the query and the document. This result suggests that, even for the dense model (which does not operate at the lexical level), lexical overlap remains a dominant signal.

The authors Ram et al. [67] conducted an experiment demonstrating that tokens shared between the query and the document tend to appear in high positions in the rankings derived from the projections of Q and D. However, documents contain many tokens, and those that are shared correspond to a small fraction of the document's tokens, leading to the following hypothesis: tokens that are more likely to appear in the query receive higher scores in D than other tokens. If this hypothesis is true, it implies that the document encoder implicitly predicts which tokens in the document are more likely to appear in queries.

To validate the hypothesis, the authors analyzed the positions (ranks) of the query and document tokens in the document projection D.

For the query q and each corresponding revalant document d, the sets of tokens composing them were obtained: $\mathcal{T}_q$ and $\mathcal{T}_d$, respectively.

		Token-level MRR in D	
		DPR	BERT (mean)
Document tokens	$\mathcal{T}_d$	3.0	0.5
Query tokens	$\mathcal{T}_q$	17.3	1.0
Shared tokens	$\mathcal{T}_d \cup \mathcal{T}_q$	26.1	1.4

Table 2.1: *Analysis of token-level MRR (in %) of document projection D into vocabulary space on the NQ development set. For the set $\mathcal{T}$, $\frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}}\frac{1}{rank_D(t)}$ is reported [67]*

Table 2.1 shows the MRR values w.r.t D for the following types of tokens: $\mathcal{T}_d$ - document tokens, $\mathcal{T}_q$ - query tokens, and $\mathcal{T}_d \cup \mathcal{T}_q$ - tokens that are in both d and q.

It is observed in Table 2.1 that the tokens shared between the query and the document $\mathcal{T}_d \cup \mathcal{T}_q$ are ranked significantly higher than the other document tokens $\mathcal{T}_d$. For example, in the DPR [29], the mean reciprocal rank (MRR) of shared tokens is 26.1, while for other document tokens, it is only 3.0. In the BERT, the MRR of shared tokens is only 1.4.

These results support the hypothesis that tokens appearing in relevant queries are ranked higher than others, and this behavior is acquired during the fine-tuning of the models.

Ram et al. [67] noted that the query encoder implements query expansion. The authors define a token as an expansion token if it does not appear in the original query q but appears in the projection Q of q and in the relevant document d (excluding stopwords and punctuation). An interesting finding is that the model tends to promote query expansion tokens to higher ranks after training, with nearly 14% of the top 5 tokens being expansion tokens. These tokens include synonyms, semantically similar tokens, and even tokens from the relevant document, suggesting the model's ability to leverage knowledge acquired during training.

Token Amnesia

The experiments indicated that tokens with high scores in the D projection (document vector projection) tend to appear in queries. However, in some cases, this prediction fails. Ram et al. [67] call this phenomenon token amnesia. To investigate the impact of this problem, the authors formulate the following question: does this failure in query prediction negatively affect retrieval?

Single-vector models struggle in out-of-domain scenarios, where sparse models like BM25 achieve better results. Ram et al. [67] demonstrated a strong correlation between token amnesia and the failure of single-vector models to retrieve the relevant document at the top of the ranking. To investigate this failure more deeply, the authors focused on a subset of a dataset where BM25 is able to retrieve the correct document

within the top 5 positions of the ranking. This subset contains the highest amount of token overlap between queries and documents, as BM25 retrieves the correct documents.

The token analysis reveals a significant correlation between token amnesia and the failures of single-vector models. The distribution of shared tokens between queries and documents indicates that single-vector models have difficulty capturing these signals.

Lexical Enrichment

Ram et al. [67] demonstrated that, due to the token amnesia phenomenon, single-vector models tend to ignore some of their input tokens. To mitigate this problem, the authors proposed a method that modifies the encoding of the dense model so that the contextualized vector $T[\text{CLS}]$ includes the ignored input tokens.

Suppose we want to enrich a document d with a token t. Lexical enrichment of a token considers t as the only token in d and seeks a dense vector s_t whose projection into the vocabulary causes t to appear at the top of the distribution D. The final vector s_t is referred to as single-token lexical enrichment and is defined by Eq. 2-41:

$$s_t = \arg\max_{\hat{s}} \log \text{MLM-Head}(\hat{s})[t] \tag{2-41}$$

To solve the optimization problem in Eq. 2-41, the Adam optimizer is used with a learning rate of 0.01 or 0.03 to rank t at the top when $\hat{s}$ is projected into the vocabulary space. Optimization is stopped when the cross-entropy loss reaches 0.1 for all tokens. Finally, whitening is applied to improve the vector.

The authors also proposed multi-token enrichment. Suppose we have an input $x = [x_1, \cdots, x_n]$ with the goal of enriching its vector representation e_x with each of its tokens x_i, $1 \leq i \leq n$, such that rare tokens are assigned higher weights (as in BM25).

The enrichment e'_x of the vector e_x is done by a weighted sum with weights of the n single-token enrichments:

$$\begin{aligned} e_x^{\text{lex}} &= \frac{1}{n} \sum_{i=1}^{n} w_{x_i} s_{x_i} \\ e'_x &= e_x + \lambda \cdot \frac{e_x^{\text{lex}}}{||e_x^{\text{lex}}||} \end{aligned} \tag{2-42}$$

In Eq. 2-42, λ is a hyperparameter chosen via cross-validation. The weights for each enrichment are obtained through the IDF of the respective token.

Relevance scores are calculated using the dot product between the enriched query vectors and the enriched document vectors. The experiments conducted demonstrate the effectiveness of the proposed method across various models, especially in zero-shot scenarios.

2.8.8 Token Pruning for ColBERT

Lassance et al. [32] noted that the big downside of ColBERT[30] is the index size, which scales linearly with the number of tokens in the collection. In order to attack this problem, they limited the number of tokens to be saved in each document using 4 token pruning techniques based on i) position (First) of the token, ii) Inverse Document Frequency (IDF) of the token, iii) special tokens, and iv) attention mechanism of the last layer.

First k tokens

A simple and effective baseline involves retaining the first k tokens of the document, based on the observation that initial tokens are more important for understanding the content. This approach leverages the inherent bias of the MS MARCO dataset[55], where the first tokens are considered the most important [26]. Moreover, in this method, Lassance et al. [32] retained punctuation tokens while selecting the first k tokens.

Top k IDF tokens

In this approach, Lassance et al. [32] retained the rarest tokens of a document, that is, the ones with the highest Inverse Document Frequency (IDF), allowing them to keep only the most discriminative words of a document.

k-Unused Tokens

Lassance et al. [32] explored the use of k-unused tokens, where k special tokens (unused tokens from the BERT vocabulary) are added at the beginning of a document, ensuring that only these tokens are retained. This method enhances model consistency by maintaining fixed token positions across all documents, potentially simplifying optimization. However, k-unused tokens has certain limitations, the method increases computational cost by enforcing a minimum document length of k tokens, and may lead to the loss of contextual information in longer documents due to truncation done with less "real-word" tokens.

Attention score

Pre-trained language models such as BERT reviewed in Section 2.3 return several key outputs: the last hidden state, which contains contextualized embeddings for each token in the input sequence; the pooler output, representing the [CLS] token embedding; a list of hidden states capturing intermediate layer representations; and attention tensors, which provide attention matrices from all transformer layers.

For instance, the last layer of attention of a document $\mathbf{A}_D^{(L)}$ is a three-dimensional tensor (number of heads, number of tokens, number of tokens'), where the first dimension is the number of different "attentions" and the last two dimensions represent the number of tokens in the input sequence (including special tokens like [CLS] and [SEP]) and the amount of attention a token t "pays" to token t', which is normalized by the softmax function so that the sum of attentions across all tokens is 1 (per document token).

The Attention score per head is computed as shown in Eq. 2-7 from Section 2.3. Let $\mathbf{A}_D^{(L)} \in \mathbb{R}^{H \times T \times T}$ denote the attention matrices from the L-th (last) transformer layer for document D, where H is the number of attention heads and T is the number of tokens. Lassance et al. [32] proposed an importance score per token which is the sum of attention paid to the token over all heads:

$$i(t)_D = \sum_{i=0}^{h} \sum_{j=0}^{|t_D|} (\mathbf{A}_D^{(L)})_{i,t,j} \tag{2-43}$$

The authors retained top-k document tokens based on attention scores derived from last layer of attention of the PLM and Eq. 2-43. They also analyzed the tokens selected by the attention mechanism and found that attention scores selects the same token at different position, leaving for future work token selection methods that take into account either the repetition problem or model the diversity of the selected tokens.

Integration and results of pruning techniques

These token pruning techniques are integrated during the training of ColBERT models because they are more effective than integration after training [32]. During and after the training, a token pruning method is added on top of ColBERT to retain top-k tokens per document and then the ColBERT's MaxSim from Eq. 2-30 is used restricted to the top-k retained tokens. Additionally, ColBERT employs BERT as its PLM, while Lassance et al. [32], used MINILM-12-384[7] as the PLM. Because of that, they investigated if the normalization and query expansion operations are helpful for all PLM, and concluded that when using MINILM instead of BERT there is no need for normalization and query expansion.

The experimental results of Lassance et al. [32] show that ColBERT indexes can be pruned up to 30% on the MS MARCO passage collection without a significant effectiviness drop, while on MS MARCO documents reveal several challenges for attention mechanism of the last layer. However, they found that the index size varies due to: the size of passages; the use or not of punctuation (First); repeated scores (attention and IDF) or because a method increases the original passage length (Unused Tokens).

[7] available at https://huggingface.co/microsoft/MiniLM-L12-H384-uncased

2.8.9 Static Pruning for Multi-Representation Dense Retrieval

Recent multi-vector models such as ColBERT [30] leverage quantized indexing techniques to generate highly compressed representations of the vectors. However, they require significant storage space for each token's vector, and longer query processing times. One of the solutions to this problem is to remove the vectors that do not contribute significantly on the retrieval effectiveness.

Acquavia et al. [1] noted that a similar approach has been investigated in the past for sparse retrieval based on inverted index data structures, named static pruning [7], where documents, terms or document-term pairs are removed from the collection or the inverted index. Therefore, they focus on adapting the IDF-based static pruning strategies to vector indexes in multi-vector models to reduce the storage overhead.

Static Pruning

Static pruning [7] is a technique used in information retrieval to reduce the size of an inverted index while maintaining efficient and effective query processing. Inverted indexes, commonly used in search engines, consist of posting lists that map unique terms to the documents in which they appear. Static pruning methods target different components of these indexes during the index construction phase and remains fixed throughout retrieval, either by removing entire terms, individual postings[8], or full documents based on predefined criteria.

One common approach is uniform pruning [1], where terms or documents are removed entirely based on global importance thresholds. Terms with low discriminative power, often measured through metrics such as IDF, may be pruned to optimize storage and retrieval efficiency. Similarly, documents that fall below a given threshold can be excluded from the inverted index.

Another strategy is term-centric pruning [1], which selectively removes postings within a term's posting list based on their relative ranking. Variants of this approach may also incorporate additional relevance signals, such as co-occurrence patterns, to refine pruning decisions. However, Acquavia et al. [1], concluded that term-centric pruning cannot be adapted to a dense multi-vector models' scenario where terms and/or documents are represented with vectors and stored in a metric index, such as FAISS [28], because the term-centric pruning relies on pruning of terms in the same posting list, but there are no obvious ways to organize multi-vectors into posting lists.

In contrast, document-centric pruning [1] determines the importance of postings within each document, removing those that contribute less to the document's overall

[8]Posting is an entry in an inverted index that records document ID and additional metadata such as term frequency and positions.

relevance. This can be achieved by analyzing term distributions within a document and across the entire collection.

Based on literature findings, Acquavia et al. [1] proposed two adaptations of static pruning strategies for vector indexes: uniform pruning and document-centric pruning.

Uniform Pruning

Uniform pruning removes all occurrences of token vectors in any document that are below predefined threshold (tokens to be removed globally) based on the rank of their global importance across the entire collection. However, some document may remain untouched. In this category, Acquavia et al. [1] implemented:

- Stopwords, here, they used a pre-defined list of stopwords from Terrier, which consist of 733 tokens, but the authors used only the 403 tokens that have an exact match with a token in the BERT tokenizer. Then removed the document vectors for tokens matching a list of stopwords
- IDF, a global tokens importance based on IDF values is computed on the MS MARCO corpus, and then the document vectors corresponding to the predefined threshold of tokens with lowest IDF are removed from the entire corpus. Acquavia et al. [1] Noted that the lowest IDF tokens are very similar to tokens that occur in a stopwords list.
- MS MARCO, which computes the IDF using the MS MARCO train queryset and then removes the vectors corresponding to the tokens with lowest IDF from the entire corpus.
- MSN, which computes the IDF using the MSN query log [8], and then removes the vectors corresponding to the predefined threshold of tokens with lowest IDF from the entire corpus.

Document-centric Pruning

Document-centric prunes all documents up to a predefined threshold, methods in this category removes a number of vectors in each document corresponding to the tokens ranked low by their global importance or random tokens. Acquavia et al. [1] implemented:

- Random doc-centric, which removes random tokens from each document. Acquavia et al. [1] sampled tokens without replacement.
- IDF doc-centric, which removes vectors that correspond to the tokens in the document with lowest IDF up to a predefined threshold.

Pruning results

The experimental results on the MS MARCO v1 passage ranking corpus presented in the work of Acquavia et al. [1] demonstrated that by removing all vectors corresponding to the 100 most frequent BERT tokens, index size is reduced by 45%, without significant degradation of nDCG@10 or MAP on the TREC 2019 or TREC 2020 querysets, and only 3% reduction in MRR on the MSMARCO Dev queryset. Moreover, they validated the generalization of the results on the TREC Covid test collection, observing a 1.3% reduction in nDCG@10 for a 38% reduction in total index size.

2.8.10 Matching Mechanisms and Token Pruning

Liu et al. [41] investigated how late interaction models compute the relevance score for query-document pairs, due to the disparate lengths of the documents, the tokens are partitioned into ten bins of equal size according to their relative position or IDF values, then two contribution metrics are defined for one document token, named indices contribution and score contribution. The indices contribution represents whether the token is selected by the max operation, while the score contribution denotes the corresponding inner product score which contributes to the final score.

The authors conducted experiment and the results showed that the computation of the relevance score heavily relies on the co-occurrence signals and the important words in documents, indicating that the words that have a higher IDF value or appear at the beginning of the documents are more important when computing the relevance score. They also found that late interaction models pay less attention to stop words. This behavior is similar to that of traditional retrieval models based on Bag of Words (BoW), where tokens with low relevance and stop words can be discarded for relevance estimation without compromising effectiveness, using weighting schemes such as IDF to determine token importance.

Based on these findings, three different token pruning strategies for documents are proposed and evaluated by Liu et al. [41], considering their positions, IDF values, and attention scores, with the goal of optimizing storage. After training models, Liu et al. [41] employed the pruning methods only during indexing to determine whether to store the token vector. All proposed pruning methods are dynamic and preserve a specific proportion of relevant tokens. Specifically, a remaining ratio denoted by α is defined, determining the percentage of tokens retained in the index. Thus, for each document, only $l' = \lfloor l \times \alpha \rfloor$ token vectors are stored, where l represents the original size of the tokenized document.

Note that Liu et al. [41] always preserved regardless of their pruning scores the special tokens positioned at the beginning of the sequence, such as [CLS] and [D]

for ColBERT, and [CLS] for COIL. These tokens play a fundamental role, as their vectors frequently encode global document information, directly contributing to retrieval effectiveness.

Token pruning strategies

First-k Pruning

The First-k method retains the initial proportion of $l' = \lfloor l \times \alpha \rfloor$ of tokens in each document. This approach is employed due to the tendency of document authors to include crucial information at the beginning of the text.

$$D' = D[1 : l'] \tag{2-44}$$

IDF-Top-k Pruning

Document tokens are pruned based on their IDF values. Liu et al. [41] sorted the document tokens in descending order of their IDF values, and only the vectors corresponding to the tokens with the highest IDF values, equivalent to $\lfloor l \times \alpha \rfloor$, are indexed.

$$\text{Top-IDF-Index} = \text{Top-}l'(IDF(d)) \tag{2-45}$$

$$D' = D[\text{Top-IDF-Index}] \tag{2-46}$$

Attention-Top-k Pruning

The Attention-Top-k method leverages attention scores as a metric to assess the importance of each token. In this approach, the self-attention matrix is computed from the interaction between the document token vectors. The sum of the columns of this matrix, which is symmetric (i.e., column sums equal row sums), represents the importance of each token. Subsequently, the top $\lfloor l \times \alpha \rfloor$ tokens with the highest attention scores are retained. Formally, the Attention-Top-k pruning is defined as:

$$A = \text{softmax}(DD^T) \cdot 1_n \tag{2-47}$$

$$\text{Top-Att-Index} = \text{Top-}l'(A) \tag{2-48}$$

$$D' = D[\text{Top-Att-Index}] \tag{2-49}$$

Query Token Pruning

Liu et al. [41] also investigated how query tokens contribute to retrieving relevant documents. Tonellotto and Macdonald [78] demonstrated that not all query vectors contribute to document ranking effectiveness. Thus, performing approximate nearest neighbor searches (Top-k ANNs) with all query tokens introduces unnecessary latency. Tonellotto and Macdonald [78] explored IDF-based pruning and found that only query vectors with the highest IDF values are effective in retrieval.

Following this perspective, Liu et al. [41] proposed a query token pruning method for ColBERT, aiming to reduce retrieval latency. The proposed pruning method is based on the self-attention matrix of query token vectors. They argue that tokens with high mean self-attention values capture the core semantics of the query and play an essential role in retrieving relevant documents. Conversely, tokens with low self-attention scores may contain unique and complementary information, making them equally relevant to retrieval. Thus, to preserve as much information as possible from the original query, the method combines the selection of tokens with the highest self-attention scores with those having the lowest scores.

Pruning formulation and results

Given a query q and document d, document token pruning methods can be expressed as follows:

$$Q = PLM(q), \quad D' = DTP(PLM(d)), \quad DTP \in \{\text{First-}\alpha, \text{IDF-Top-}\alpha, \text{Attention-Top-}\alpha\} \tag{2-50}$$

$$s(q,d) = \text{Match}(Q, D') \tag{2-51}$$

where Q is a set of vectors, PLM is an encoder based on pre-trained language model, D' is a set of vectors after pruning, DTP is a pruning method based on position, IDF or Attention scores. $Match$ is an matching model that compute relevance score of a document d w.r.t a query q.

The experimental results of Liu et al. [41] showed that on the MS MARCO and BEIR, storage overhead can be reduced with little performance loss when use late-interaction models such as COIL and ColBERTv2. Liu et al. [41] also investigated the inference time and found that the query token pruning methods can reduce the retrieval latency with little loss of performance for ColBERT on the MS MARCO.

CHAPTER 3

Novel pruning methods

In this chapter we introduce two novel pruning methods for multi-vector bi-encoder retrievers. The first one is the **TF-IDF** method described in Section 3.1. The second method is the **MMTR**, described in Section 3.2.

3.1 The TF-IDF method

TF-IDF (Term Frequency-Inverse Document Frequency) is a classic weighting scheme designed to evaluate the importance of a term to a document and to an entire collection. Its computation is described in Eq. 3-1:

$$\text{TF-IDF}(t) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \cdot \log \frac{N}{1 + df(t)}, \tag{3-1}$$

where the first term, referred as *term frequency* (TF), corresponds to the relative frequency of token t in document d and the second term corresponds to the *inverse document frequecy* (IDF), already described in Eq. 2-2. The pruning method consists in ranking the tokens of a document in descending order of their TF-IDF values and keeping only the vectors corresponding to the top k terms.

3.2 The MLM Max with Token Reordering - MMTR

The MMTR method consists of three steps: a) the embeddings projection, b) pooling, and c) token sorting. We detail these steps in the following sections.

3.2.1 Embedding projection

We follow Ram et al. [67] and project each token of a document d into the BERT [12] wordpiece vocabulary space (V_{BERT}) using a Masked Language Modeling (MLM) head [67]. Since PLAID is based on BERT, we reuse the BERT pre-trained MLM head weights to perform these projections. Let $= \{v_1, v_2, \cdots v_{|d|}\}$ be the set of vectors

output by the PLAID for tokens of a document d and let $W_{\text{MLM}} \in \mathbb{R}^{d \times |V_{\text{BERT}}|}$ be the matrix containing BERT pre-trained MLM head weights. The projection P_i of a token vector v_i is computed by Eq. 3-2. P_i represents the probability distribution over the 30,522 tokens of the vocabulary, generated for the token vector v_i.

$$P_i = softmax(v_i \cdot W_{MLM}) \tag{3-2}$$

3.2.2 Pooling strategy

The goal of the pooling step is to aggregate the projections produced in the first step into a single representation. Let $\mathcal{P}_d = \{P_1, P_2, \cdots P_{|d|}\}$, be the set of projections of token vectors corresponding to document d. Each projection $P_i = \{p_{i,1}, p_{i,2}, \cdots p_{i,V_{|\text{BERT}|}}\}$, is composed of probability values $p_{i,j}, 1 \leq j \leq |V_{\text{BERT}}|$. We apply the max-pooling strategy over the projections to obtain a single aggregated score s_j for each token j in BERT's vocabulary, as shown in Eq. 3-3

$$s_j = \max_{1 \leq i \leq |d|} (p_{i,j}) \tag{3-3}$$

Thus, we have a list $\mathcal{L}$ of scores, as defined in Eq. 3-4.

$$\mathcal{L} = [s_1, s_2, \cdots s_{|\text{BERT}|}] \tag{3-4}$$

3.2.3 Token sorting

Note that list $\mathcal{L}$ contains a score for each vocabulary token, including tokens not present in the document d. Since our objective is to prune tokens occurring in documents, we first create a representation $\mathcal{L}_d$ for a document d containing $|d|$ pairs, where the first component t of a pair is a token occurring in d and the second component is t's aggregate score s_t, as shown in Eq. 3-5

$$\mathcal{L}_d = [(t, s_t) | t = t_i \in d \text{ and } s_t \text{ is t's score, computed by Eq. } 3-3] \tag{3-5}$$

We then obtain a ranking $\mathcal{R}_d$ of pairs in $\mathcal{L}_d$ by sorting the pairs in $\mathcal{L}_d$ in descending order of their token aggregated scores. This ranking reflects the relative importance of each token for the document d.

A token t may occur multiple times in a same document d. Since the different embeddings of token t are similar among themselves, their importance score also tend to be similar. This imply that the different occurrences of token t tend to group near each

other in $\mathcal{R}_d$. The right column of Table 3.1 illustrates this behavior for a document d shown in the left column of the Table [1].

Table 3.1: *The token grouping Problem. Tokens written in red color in $\mathcal{R}_d$ represent repeated tokens and their grouping near each other.*

Document d	Token Ranking $\mathcal{R}_d$
What then is the interest of the animal? If releasing these animals into the wild would kill them then surely it is humane to put them down after the experiment. It must also be remembered that the interest of the animal is not the main and is outweighed by the benefits to humans. [5]	['interest', 'interest', 'down', 'after', 'put', 'wild', 'not', 'humane', 'of', 'of', 'if', 'experiment', 'into', 'animal', 'animal', 'then', 'then', 'remembered', 'releasing', 'main', 'would', 'must', 'surely', 'animals', 'kill', 'to', 'to', '##weig', 'be', 'benefits', 'out', 'the', 'the', 'the', 'the', 'the', 'the', 'the', 'the', 'it', 'it', 'humans', 'is', 'is', 'is', 'is', 'them', 'them', 'what', 'also', '5', 'and', '##hed', 'that', 'by', '?', '.', '.', 'these', '[', ']', '[D]', '[CLS]', '[SEP]']

The grouping of different occurrences of the same token in the top of the ranking $\mathcal{R}_d$ diminishes the effectiveness of retriever using pruned index. We confirm this observation in our ablation analysis in Section 5.1.4. Particularly, token grouping is more inconvenient in settings with a low remaining ratio, as it prevents the inclusion of different but also important token vectors in the index.

To mitigate the token grouping problem, we propose a re-ordering of the ranking $\mathcal{R}_d$ for a document d by prioritizing the first occurrence of each token in $\mathcal{R}_d$. Subsequent occurrences of a same token are recursively pushed down in the new rank which we refer as $\mathcal{R}_d^p$ *Prioritized ranking of tokens in d*. This re-ordering is described in Algorithm 3.1.

The right column of Table 3.2 shows the result ($\mathcal{R}_d^p$) of Algorithm 3.1 after applied to the input $\mathcal{R}_d$ in the left column of the Table. Notice that Algorithm 3.1 positions the first occurrence of all tokens near the top of $\mathcal{R}_d^p$, in descending order of their aggregated score. Next, it positions the second occurrences of repeated tokens (if they exist) in descending order of their aggregated score, and so on.

[1] We show only the terms in the pairs in $\mathcal{R}_d$ (right column of Table 3.1) to save space.

Algorithm 3.1: Obtaining a new ranking $\mathcal{R}_d^p$ from $\mathcal{R}_d$ by prioritizing the first occurrence of each token

Input: $\mathcal{R}_d$ - the ranking of tokens in d by aggregated score
Output: $\mathcal{R}_d^p$ - the new ranking prioritizing the first occurrences of tokens in d

```
Initialize R_d^p ← {}
Initialize seen ← {}
while R_d ≠ {} do
    Initialize unique_part ← {}
    Initialize repeated_part ← {}
    foreach each pair (t, s_t) in R_d do
        if t ∉ seen then
            Insert (t, s_t) at the end of unique_part
            Add t to seen
        else
            Insert (t, s_t) at the end of repeated_part
    R_d^p ← Concatenate (R_d^p, unique_part)
    R_d ← repeated_part
```

Table 3.2: *Output of Algorithm 3.1 is shown in right column ($\mathcal{R}_d^p$), while the input of the Algorithm ($\mathcal{R}_d$ is shown in left column . Tokens in red color are repeated tokens.*

Token Ranking	Token Prioritization
['interest', 'interest', 'down', 'after', 'put', 'wild', 'not', 'humane', 'of', 'of', 'if', 'experiment', 'into', 'animal', 'animal', 'then', 'then', 'remembered', 'releasing', 'main', 'would', 'must', 'surely', 'animals', 'kill', 'to', 'to', '##weig', 'be', 'benefits', 'out', 'the', 'the', 'the', 'the', 'the', 'the', 'the', 'the', 'it', 'it', 'humans', 'is', 'is', 'is', 'is', 'them', 'them', 'what', 'also', '5', 'and', '##hed', 'that', 'by', '?', '.', '.', 'these', '[', ']', '[D]', '[CLS]', '[SEP]']	['interest', 'down', 'after', 'put', 'wild', 'not', 'humane', 'of', 'if', 'experiment', 'into', 'animal', 'then', 'remembered', 'releasing', 'main', 'would', 'must', 'surely', 'animals', 'kill', 'to', '##weig', 'be', 'benefits', 'out', 'the', 'it', 'humans', 'is', 'them', 'what', 'also', '5', 'and', '##hed', 'that', 'by', '?', '.', 'these', '[', ']', '[D]', '[CLS]', '[SEP]', 'interest', 'of', 'animal', 'then', 'to', 'the', 'it', 'is', 'them', '.', 'the', 'is', 'the', 'is', 'the', 'the', 'the', 'the']

CHAPTER 4

Experimental Setup

4.1 Datasets

Many recent work train transformer-based retrieval models on large datasets such as MS MARCO (MAchine Reading COmprehension Dataset) [55], which focus on passage retrieval given a query. The retrieval model is then evaluated on the same dataset, which is called an in-domain dataset[1]. However, it is unclear how well the trained retrieval model will perform for other text domains or textual retrieval tasks [77], which is called an out-of-domain dataset[2]. The BEIR (Benchmarking-IR) [77], a robust and heterogeneous evaluation benchmark for information retrieval, is proposed to fill this gap, aiming to provide a zero-shot evaluation benchmark for all diverse retrieval tasks, where a majority of datasets contain binary relevancy judgements, i.e. relevant or non-relevant, and a few contain fine-grained relevancy judgements.

We follow prior work by Liu et al. [41] and consider in-domain and out-of-domain datasets for the evaluation of pruning methods.

4.1.1 In-domain Evaluation

MS MARCO Passage

We evaluate in-domain retrieval effectiveness and efficiency on MS MARCO Passage Dev Small [55], which contains 8,841,823 passages extracted from 3,563,535 web documents retrieved by a state-of-the-art passage retrieval system at Bing, and 6,980 anonymized queries issued through Bing or Cortana log, with an average of one relevant passage per query, but a query in the MS MARCO dataset may have multiple relevant passages or non-relevant passage at all. The authors included these queries with non-relevant passage in dataset because they believe that the ability to recognize insufficient

[1] In-domain datasets correspond to those used for training PLAID, maintaining a similar distribution and characteristics.

[2] Out-of-domain datasets introduce unseen contexts, differing from the training data, and serve to evaluate the model's generalization ability.

Table 4.1: *The judgments of a list of passages from the full collection on a four-point scale.*

Score	Judgment Level	Description
3	Perfectly Relevant	The passage is fully dedicated to the query and contains the exact answer.
2	Highly Relevant	The passage provides an answer, but it may be unclear or mixed with extraneous information.
1	Related	The passage is related to the query but does not provide an answer.
0	Irrelevant	The passage is unrelated to the query.

(or conflicting) information that makes a query have no relevant passage is important to develop for a model.

Given an average set of 10 passages per query retrieved by Bing, human editors conducted binary relevance judgments, where passages that contain useful and necessary information w.r.t a query were annotated as relevant by setting score to 1, while the rest were annotated as non-relevant by setting score to 0.

TREC DL 2019

We also evaluate on the TREC DL 2019 (Text REtrieval Conference 2019 Deep Learning) [9], using passage retrieval task, which contains 200 queries, 43 of which were judged by NIST assessors. The passage task is similar to the MS MARCO passage ranking, but with a new test set in the TREC version with more comprehensive labeling, meaning that the passage corpus is the same as in MS MARCO passage [55].

The passage retrieval task consists of two subtasks [9]: full retrieval, where participants retrieve a ranked list of passages from the full collection w.r.t a query, submitting up to 1000 passages per query; and top-1000 reranking, where given a query, participants rerank a given predefined set of 1000 BM25-retrieved passages per query. NIST's pooling method was used for judging from both subtasks. To ensure comprehensive set of relevant results, classifiers and high-relevance queries were used to identify additional passages for judging. The judgments were made on a four-point scale, shown in Table 4.1.

4.1.2 Out-of-domain Evaluation

We evaluate effectiveness on BEIR benchmarks [77], consisting of 18 IR datasets related to search and semantic tasks across various domains, ranging from Wikipedia, scientific publications, Twitter, news, to online user communities, for comparison and evaluation of model generalization. The authors included 18 English zero-shot evaluation

datasets from 9 heterogeneous retrieval, including fact checking, citation prediction, duplicate question retrieval, argument retrieval, question answering, bio-medical IR, and entity retrieval.

We conduct a zero-shot evaluation on 13 publicly-available datasets, shown in Table 4.2. We refer to Thakur et al. [77] for a more detailed description of each of the datasets.

Table 4.2: *BEIR dataset information.*

Task	Domain	Dataset	Test Split	
			# Query	# Corpus
Bio-medical	Bio-medical	TREC-COVID	50	171,332
information retrieval	Bio-medical	NFCorpus	323	3,633
Question answering	Wikipedia	NQ	3,452	2,681,468
	Wikipedia	HotpotQA	7,405	5,233,329
	Finance	FiQA-2018	648	57,638
Argument retrieval	Misc	ArguAna	1,406	8,674
	Misc	Touché-2020	49	382,545
Duplicate-question retrieval	Quora	Quora	10,000	522,931
Entity-retrieval	Wikipedia	DBPedia	400	4,635,922
Citation-prediction	Scientific	SCIDOCS	1,000	25,657
Fact checking	Wikipedia	FEVER	6,666	5,416,568
	Wikipedia	Climate-FEVER	1,535	5,416,593
	Scientific	SciFact	300	5,183

4.2 IR Evaluation Metrics

An important goal of IR Evaluation Metrics is to compare the performance of different types of methods across various datasets. These metrics are computed from relevance judgements (also called qrels) to quantify the effectiveness of a method.

During evaluation, qrels, which consist of a set of (query, document, relevance label) triples, are used alongside a ranked list of retrieved results for all test queries—referred to as a run. These are then fed into an evaluation program (e.g., ir_measures[3]), which automatically computes evaluation metrics per query. Finally, each metric is aggregated across all queries to obtain a global measure of the method's effectiveness.

[3] https://ir-measur.es/en/latest/

Depending upon the nature and requirements of MS MARCO Passage [55], TREC DL 2019 [9] and BEIR benchmarks [77], we report three metrics: the MRR; nDCG; and R. We show in Section 4.1 that MS MARCO Passage has binary relevance judgements making binary rank-aware metrics such as MRR and MAP suitable for this task, but we report only the official MRR@10 metric. However, these metrics fail to evaluate tasks with graded relevance judgements, where nDCG provides a good balance suitable for both tasks involving binary and graded relevance judgements [77]. Therefore, for TREC DL 2019, which have graded relevance judgements, and for BEIR benchmarks, which have both binary and graded relevance judgements, we report the official metric nDCG@10.

We also follow prior work by Liu et al. [41] and report the R@100 metric for MS MARCO Passage and TREC DL 2019 instead of the usually used R@1000, because the latency of reranking 1000 candidates is unacceptable for the online dense retrieval systems.

In addition to effectiveness, we also assess the efficiency of each method in terms of the index size and query processing time aiming at analyzing the effectiveness-efficiency trade-offs in all methods. The time metric is the average time per query in milliseconds measured in the retrieval of documents of each model or pruning method.

4.3 Statistical Significance Tests

Statistical significance tests is widely accepted as a means to found out whether there is a significant difference between the effectiveness of models [79, 66]. Student t-test, also called t-test, is the most popular choice among IR researchers.

Urbano et al. [79] followed a simulation approach that allowed they to evaluate statistical significance tests with unlimited IR-like data and under full control of the null hypothesis. Their findings indicate that the t-test is simpler and remarkably robust to sample size, so it becomes their top recommendation. However, the issue of statistical tests for comparisons of more models on multiple datasets has been ignored [11].

Demšar [11] studied statistical tests that can be used for comparing two or more models on multiple datasets. Based on the experimental results, the author recommended the Friedman test with the corresponding post-hoc tests such as Hommel for comparison of more models over multiple datasets. We refer to Urbano et al. [79], Demšar [11], and Rainio et al. [66] for a more detailed explanation of t-test, Friedman test, and Hommel post-hoc test.

According to Rainio et al. [66] the process of statistical testing consist of formulating a null hypothesis $H0$, witch states that all models perform the same and the observed differences are merely random, choose some level of significance $\alpha \in (0, 1)$,

and define a suitable test statistic Z with a known probability distribution $P(Z|H0)$ under the null hypothesis.

We used Statistical Tests for Data Science (StaTDS)[4] [46] to compute the probability, called $p-value$, when we get a $p-value$ less than α, the null hypothesis is rejected and the difference is considered statistically significant between the models.

For MS MARCO Passage and TREC DL 2019, we performed statistical significance t-tests with $p-value < 0.05$. Statistically significant differences are indicated with the symbol † for MMTR, and the symbol ‡ for TF-IDF. Note that the aforementioned datasets use different metrics, MRR and nDCG, so we cannot use the Friedman test with a post-hoc test, as recommended by Demšar [11] and Rainio et al. [66].

We conducted the Friedman test for the BEIR benchmarks because it includes 13 datasets and uses the same evaluation metric across them. We analyzed the runs (retrieval results of the models) by representing each run as a vector of 13 nDCG@10 scores. In this vector space, two runs are statistically similar on the same queries if their nDCG vectors are similar. When the $p-value$ obtained is less than $\alpha = 0.05$ in a comparison between models across these datasets, we can reject the null hypothesis and conclude that there are statistically significant differences between models.

In this case, we proceed with an Hommel post-hoc test to identify which models actually differ. In this setting, we also used a confidence level of significance of $p-value < 0.05$. Then we indicate significant differences with the symbol † for MMTR, and the symbol ‡ for TF-IDF.

4.4 Baselines

We compare our two proposed pruning methods TF-IDF and MMTR against three other pruning methods investigated in literature: Pruning by token position, IDF, and attention scores. These pruning criterion was used by Lassance et al. [32] and Liu et al. [41] for pruning the multi-vectors of each document[5]. In this work, we take the baseline pruning methods from Liu et al. [41]. For each dataset, we applied each of the pruning methods to the token vectors produced by PLAID. We also compare the performance of PLAID without pruning to assess the impact of pruning on the original model.

[4] https://statds.readthedocs.io/en/latest/

[5] In fact Lassance et al. [32] also investigated the criterion of inserting k unused token at the beginning of each document and keep only these tokens. However, the authors themselves recognized that this criterion is problematic. This criterion was not considered by Liu et al. [41] and will also not be considered in our work.

4.4.1 Pruning by token position

Here, we retain only the tokens appearing in the k first positions of the documents.

4.4.2 Pruning by IDF

In this approach, we retain the k tokens in each document with the highest Inverse Document Frequency (IDF) scores, computed according to Eq. 4-1

$$IDF(t) = \log \frac{N}{1+df(t)}, \tag{4-1}$$

where N is the number of documents in the dataset, and $df(t)$ is the number of documents containing at least one occurrence of token t.

4.4.3 Pruning by attention scores

We use as baseline and describe in more detail the attention criterion used by Liu et al[6] [41]. Let $D \in \mathbb{R}^{|d| \times c}$ be the matrix formed by the sequence of vectors $[v_1, v_2, \ldots, v_{|d|}]$, each with dimension c, where each v_i is the vector corresponding to token d_i in the document d (i.e., $d = [d_1, d_2, \ldots, d_{|d|}]$). We obtain matrix B according to Eq. 4-2

$$B = D \cdot D^T \tag{4-2}$$

Next, the attention matrix A is obtained by normalizing B, using the softmax function, as shown in Eq. 4-3

$$A = softmax(B) \tag{4-3}$$

The attention matrix A captures the pairwise attention scores between all tokens in the document d. Next, the importance score $s(t_i)$ for each token $t_i \in d$ is obtained by aggregating the attention values between t_i and all other token in document d as shown in Eq. 4-4

$$s(t_i) = \sum_{j=1}^{|d|} A_{ij} \tag{4-4}$$

Finally, we sort tokens in d in descending order of their scores and index only the top-k token with greater scores.

[6] It differs from the attention criterion used by Lassance et al. [32], which uses scores of the attention heads of the last layer of attention of the PLM.

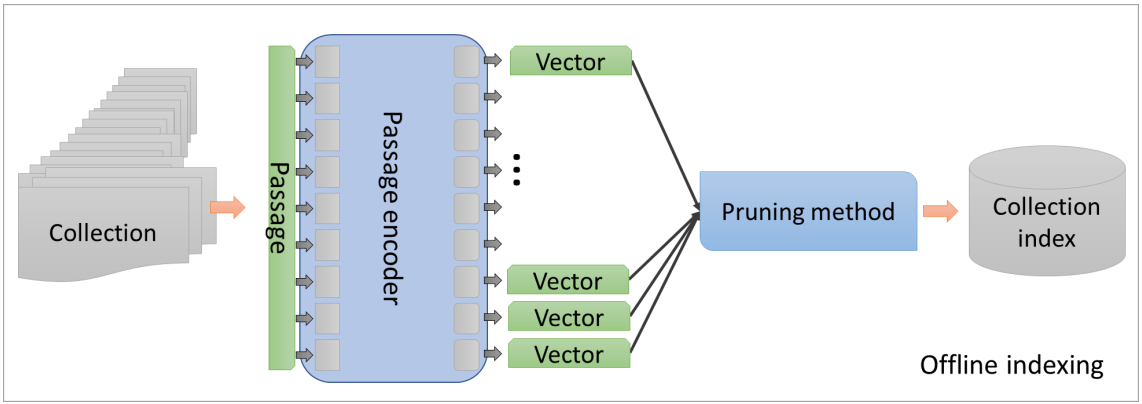


Figure 4.1: *The pipeline of indexing process. This figure describes the pipeline of the indexing process of PLAID, when static pruning is used. Each passage is fed to the passage encoder. The encoder produces a vector for each passage token, including the [CLS] and [SEP] token. We then apply a pruning method to reduce the number of passage token vectors. Next, the remaining passage token vectors are compressed by PLAID and stored in the index.*

4.5 Pruning Pipeline

Our indexing process follows the pipeline shown in Figure 4.1. We integrate the pruning method after the passage encoding process, at the encoder output tokens during indexing, which occurs offline. We adopt this strategy because we determine the importance of tokens in one of our pruning methods based on the output vectors of the encoder. Implementing dual encoding - one to evaluate the importance of the tokens and another to encode the top k tokens - would result in a significant discrepancy between the vectors from the two phases, as well as a substantial loss of context.

Given the collection of passages C, we process each passage using the passage encoder to generate contextualized token vectors. We then apply the pruning method to the vectors of each passage, evaluating the relative importance of each token vector. Based on this evaluation, we select and index the top-k most important contextualized token vectors.

4.6 Experimental Setup

We implement baselines methods as described by Liu et al. [41]. We conducted the experiments on a machine equipped with an A100-SXM4-80GB GPU. We use the PLAID source code provided by authors,[7] which is initialized from the ColBERTv2

[7] https://github.com/stanford-futuredata/ColBERT

checkpoint.[8] We modify the PLAID source code to support pruning methods. Then we follow default PLAID configurations for the indexing, while employing pruning methods, and retrieval. However, we follow Santhanam et al. [74] and set the maximum passage size to 300 tokens on BEIR indexing. For queries, we set the maximum query size to 300 tokens for the ArguAna dataset, due to the presence of long queries, and 64 tokens for the Climate-FEVER dataset, which also contains lengthy queries. We release our modified PLAID code and scripts to run our experiments.

We use IR Datasets[9] [51] to acquire, manage, and execute typical operations over datasets used in IR, and IR Measures [48] to calculate all the desired metrics, simplifying the evaluation process. For statistical significance tests, we use StaTDS [46] to analyze, test, and compare our proposed pruning methods against other methods.

[8] https://downloads.cs.stanford.edu/nlp/data/colbert/colbertv2/colbertv2.0.tar.gz
[9] https://ir-datasets.com/index.html

CHAPTER 5

Results and Discussion

5.1 Experimental Results

We show the results of the experiments conducted on MS MARCO Passage and TREC DL 19 in Table 5.1, while we present those for BEIR in Tables 5.3, 5.4 and 5.5. We evaluated the effectiveness of the model with various remaining ratios: 100%, 75%, 50%, and 25%. We apply the last three remaining ratios to the pruning methods investigated in this study, enabling a comparative analysis of effectiveness versus efficiency in different pruning scenarios.

5.1.1 Results on MS MARCO Passage and TREC DL 19

Table 5.1 presents the results for these datasets. A remaining ratio of 25% leads to a noticeable drop in effectiveness across both datasets. For instance, on the MS MARCO dataset, the MMTR method, despite statistically outperforming the other methods, shows a 14.86% decrease in MRR@10 and a 7.31% decrease in R@100 compared to the original, unpruned PLAID model.

On the TREC DL 2019 dataset, with a remaining ratio of 25%, MMTR results in a 4.69% reduction in nDCG@10 and an 11.88% drop in R@100. Both MMTR and TF-IDF outperform the First, IDF, and Attention methods in terms of R@100 across both datasets. However, all methods show similar performance in nDCG@10 on the TREC DL 2019 dataset.

At a 50% remaining ratio, half of the token vectors are pruned, leading to only a slight decrease in retrieval performance. On the MS MARCO dataset, MMTR continues to be the top-performing method, showing statistically significant improvements over the others—except for the IDF method in R@100, where the results are statistically indistinguishable. Compared to the unpruned PLAID baseline, MMTR shows only a 2.26% drop in MRR@10 and a 1.09% drop in R@100 on MS MARCO, and a 0.4% and 1.74% drop in nDCG@10 and R@100, respectively, on TREC DL 2019.

Table 5.1: *Results of experiments on MS MARCO Passage and TREC DL 19 with 100%, 75%, 50%, and 25% remaining ratios. † and ‡ indicate statistical significance of MMTR and TF-IDF, respectively, compared to the baseline pruning methods using the paired t-test ($p-value \leq 0.05$). Bold denotes the best method for that metric.*

Method	MS MARCO Passage		TREC DL 19	
	MRR@10	R@100	nDCG@10	R@100
Keep all embeddings of the passage				
PLAID	0.397	0.916	0.745	0.572
Keep 25% of the passage embeddings				
First	0.322 †	0.776 † ‡	0.694	0.413 † ‡
IDF	0.286 † ‡	0.811 † ‡	0.664	0.427 † ‡
TF-IDF	0.321 †	0.843 †	0.684	0.480
Attention	0.316 †	0.767 † ‡	0.688	0.409 † ‡
MMTR	**0.338** ‡	**0.849** ‡	**0.710**	**0.504**
Keep 50% of the passage embeddings				
First	0.376 †	0.866 † ‡	0.735	0.505 † ‡
IDF	0.381 †	0.904	0.719	0.561
TF-IDF	0.380 †	0.902 †	0.730	**0.563**
Attention	0.375 †	0.866 † ‡	0.730	0.508 † ‡
MMTR	**0.388** ‡	**0.906** ‡	**0.742**	0.562
Keep 75% of the passage embeddings				
First	0.390 †	0.901 † ‡	0.738	0.546 † ‡
IDF	0.393	0.914	**0.746**	0.566 ‡
TF-IDF	0.394	0.913	0.743	0.561 †
Attention	0.392	0.901 † ‡	0.734	0.551 † ‡
MMTR	**0.396**	**0.914**	0.742	**0.568** ‡

At a 75% remaining ratio, MMTR results in minimal performance loss—just 0.25% in MRR@10 and 0.21% in R@100 on MS MARCO, and 0.4% in nDCG@10 and 0.7% in R@100 on TREC DL 2019—when compared to the unpruned PLAID model. However, retaining 75% of the vectors provides only modest gains in index size and query latency, as discussed in Section 5.1.3. At this pruning level, MMTR, TF-IDF, and IDF perform comparably on MS MARCO, while the First and Attention methods show significantly weaker performance in R@100 on both MS MARCO and TREC DL 2019.

It is important to highlight some observations for these two datasets:

- MMTR is statistically the best-performing method on the MS MARCO dataset for

the first two remaining ratios, particularly at the 50% level.

- The First and Attention methods consistently underperform compared to MMTR and TF-IDF in terms of recall, with statistically significant differences in both datasets. This is a notable drawback when these methods are used with PLAID as the first-stage retriever.
- No single method outperforms the others in nDCG@10 on TREC DL 2019. All methods result in very minimal effectiveness loss (less than 2%) at 50% and 75% remaining ratios, when compared to the unpruned PLAID.
- Overall, TF-IDF and IDF perform nearly equally on both datasets at the 50% and 75% remaining ratios, with IDF showing a slight advantage in R@100 only on TREC DL 2019 at the 75% level.

5.1.2 Results on BEIR benchmark

We used the Friedman test to determine whether there are significant differences in the rankings of the pruning methods at each remaining ratio—25%, 50%, and 75% of the embeddings. The average rank for each method at each ratio is presented in the final rows of Tables 5.3, 5.4, and 5.5. The Friedman test rejected the null hypothesis that the average ranks of the methods are statistically equivalent across all remaining ratios.

Following these rejections, we conducted a post-hoc analysis as recommended by Demšar et al. [11] and Garcia et al. [21]. The Hommel post-hoc test was used to confirm whether the differences between the average ranks of each pair of methods are statistically significant, with results shown in Table 5.2.

Table 5.2: *Statistical differences (✓) and no statistical differences (×) between methods for different remaining ratios.*

Pairwise comparison	25% of the vectors	50% of the vectors	75% of the vectors
First vs TF-IDF	✓	✓	×
First vs MMTR	✓	✓	✓
First vs IDF	×	✓	×
First vs Attention	×	×	×
IDF vs Attention	✓	✓	×
IDF vs TF-IDF	×	×	×
IDF vs MMTR	×	×	×
TF-IDF vs Attention	✓	✓	×
TF-IDF vs MMTR	×	×	×
Attention vs MMTR	✓	✓	✓

We present the results for the 25% remaining ratio in Table 5.3. Based on the statistical analysis of the average ranks (see the first column of Table 5.2), we identify

Table 5.3: *NDCG@10 results for the BEIR Benchmark when we keep 25% of the passage embedings. † and ‡ indicate statistical difference from MMTR and TF-IDF, respectively, according to the Hommel post-hoc test ($p-value \leq 0.05$). Bold denotes the best NDCG@10 value among pruning methods for a given dataset.*

	TF-IDF	MMTR	IDF	First	Attention	PLAID
DBPedia	0.392	**0.405**	0.353	0.360 † ‡	0.254 †‡	0.442
FiQA-2018	**0.314**	0.307	0.307	0.252 †‡	0.233 †‡	0.355
Natural Questions	0.409	**0.428**	0.400	0.362 †‡	0.299 †‡	0.491
HotpotQA	0.541	**0.560**	0.537	0.479 †‡	0.327 †‡	0.667
NFCorpus	0.324	**0.325**	0.308	0.289 †‡	0.247 †‡	0.334
TREC-COVID	0.625	**0.652**	0.501	0.573 †‡	0.533 †‡	0.628
Touché-2020	**0.337**	0.332	0.324	0.318 †‡	0.286 †‡	0.351
ArguAna	0.270	0.258	0.272	**0.279** †‡	0.253 †‡	0.316
Climate-Fever	**0.177**	0.165	0.169	0.147 †‡	0.154 †‡	0.175
Fever	0.711	**0.746**	0.672	0.683 †‡	0.556 †‡	0769
Quora	0.666	0.636	**0.669**	0.588 †‡	0.603 †‡	0.854
SCIDOCS	**0.135**	0.131	0.132	0.121 †‡	0.103 †‡	0.144
Scifact	0.616	**0.628**	0.560	0.455 †‡	0.532 †‡	0.669
Rank Pos.	1.8	1.9	2.9	3.8	4.7	–

two statistically distinct groups of methods: $S_1 = \{$TF-IDF, MMTR$\}$ and $S_2 = \{$First, Attention$\}$. The methods in S_1 significantly outperform those in S_2, while there is no statistically significant difference between the methods within each group. The placement of IDF is inconclusive, as it does not show a statistically significant difference from either group.

It's important to note that, unlike the in-domain datasets, the performance losses on out-of-domain datasets show much greater variability—ranging from 2.7% to 22%.[1]

Surprisingly, even when retaining only 25% of the vectors, two datasets actually show performance gains: TREC-COVID sees a 3.8% improvement with MMTR over PLAID, and Climate-Fever shows a 1.14% improvement with TF-IDF over PLAID.

Table 5.4 presents the results for the 50% remaining ratio. The Hommel test reveals a clear distinction between two groups of methods: $S_1 = \{$TF-IDF, MMTR, IDF$\}$ and $S_2 = \{$First, Attention$\}$. When pruning 50% of PLAID's vectors, the differences in nDCG@10 between the top three pruning methods and the unpruned PLAID model decrease significantly. The largest observed drop was 6.6%, occurring in the ArguAna

[1] We calculate the loss as the difference in performance between the best pruning method (either TF-IDF or MMTR) for each dataset and the unpruned PLAID model.

Table 5.4: *NDCG@10 results for the BEIR Benchmark when we keep 50% of the passage embedings. † and ‡ indicate statistical difference from MMTR and TF-IDF, respectively, according to the Hommel post-hoc test ($p-value \leq 0.05$). Bold denotes the best NDCG@10 value among pruning methods for a given dataset.*

	TF-IDF	MMTR	IDF	First	Attention	PLAID
DBPedia	0.436	**0.438**	0.435	0.401 †‡	0.352 †‡	0.442
FiQA-2018	**0.352**	0.351	0.350	0.311 †‡	0.292 †‡	0.355
Natural Questions	0.477	**0.479**	0.476	0.422 †‡	0.383 †‡	0.491
HotpotQA	**0.652**	**0.652**	**0.652**	0.559 †‡	0.472 †‡	0.667
NFCorpus	0.333	**0.334**	0.328	0.315 †‡	0.312 †‡	0.334
TREC-COVID	**0.651**	0.624	0.642	0.619 †‡	0.634 †‡	0.628
Touché-2020	0.336	**0.344**	0.334	0.331 †‡	0.325 †‡	0.351
ArguAna	0.294	0.293	**0.295**	0.288 †‡	0.290 †‡	0.316
Climate-Fever	0.176	0.173	**0.178**	0.154 †‡	0.167 †‡	0.175
Fever	0.769	**0.773**	0.768	0.713 †‡	0.652 †‡	0.769
Quora	0.833	0.787	**0.834**	0.653 †‡	0.693 †‡	0.854
SCIDOCS	**0.142**	0.140	0.140	0.137 †‡	0.122 †‡	0.144
Scifact	0.663	**0.671**	0.663	0.593 †‡	0.624 †‡	0.669
Rank Pos.	1.8	2.0	2.3	4.4	4.5	–

dataset, where TF-IDF (one of the top three pruning methods for this dataset) underperformed compared to unpruned PLAID. On average, the loss in effectiveness was just 0.97%.

Notably, at the 50% remaining ratio, improvements over PLAID were observed in four datasets –TREC-COVID, Climate-Fever, FEVER, and SciFact – with two of these gains resulting from the MMTR method. The largest observed improvement was a 3.7% gain over PLAID, achieved by the TF-IDF method.

Table 5.5 presents the nDCG@10 results for the 75% remaining ratio. The Friedman and Hommel tests indicate that MMTR continues to outperform First and Attention. However, there is not enough statistical evidence to clearly distinguish between TF-IDF, IDF, and MMTR, nor to confirm significant differences between TF-IDF, IDF, and the First and Attention methods.

At this higher remaining ratio, the effectiveness losses are even smaller, ranging from 0.2% to 2.56%. Additionally, there are ties or slight improvements over PLAID in 8 out of the 13 datasets[2] with gains ranging from 0% to 1.1%. MMTR slightly outperforms

[2]DBPedia, FiQA-2008, TREC-COVID, Touché-2020, Climate-Fever, Fever, Quora and Scifact.

Table 5.5: *NDCG@10 results for the BEIR Benchmark when we keep 75% of the passage embedings. † and ‡ indicate statistical difference from MMTR and TF-IDF, respectively, according to the Hommel post-hoc test ($p-value \leq 0.05$). Bold denotes the best NDCG@10 value among pruning methods for a given dataset.*

	MMTR	TF-IDF	IDF	First	Attention	PLAID
DBPedia	**0.447**	0.441	0.441	0.429 †	0.402 †	0.442
FiQA-2018	**0.357**	0.353	0.353	0.344 †	0.334 †	0.355
Natural Questions	**0.489**	0.485	0.484	0.461 †	0.442 †	0.491
HotpotQA	**0.665**	**0.665**	0.664	0.623 †	0.577 †	0.667
NFCorpus	**0.331**	0.329	0.330	0.321 †	0.328 †	0.334
TREC-COVID	0.622	0.603	0.611	0.626 †	**0.653** †	0.628
Touché-2020	0.342	0.331	0.335	0.340 †	**0.354** †	0.351
ArguAna	0.311	**0.315**	0.313	0.307 †	0.310 †	0.316
Climate-Fever	**0.175**	**0.175**	**0.175**	0.170 †	0.173 †	0.175
Fever	**0.771**	0.769	0.769	0.741 †	0.715 †	0.769
Quora	0.852	**0.854**	0.853	0.765 †	0.762 †	0.854
SCIDOCS	**0.142**	**0.142**	**0.142**	0.141 †	0.136 †	0.144
Scifact	**0.673**	0.670	0.672	0.637 †	0.664 †	0.669
Rank. Pos.	1.7	2.5	2.6	4.1	4.1	–

PLAID in 4 of these 8 datasets.

5.1.3 Index Size and Query Processing Latency Analysis

In this section, efficiency is assessed based on two factors: index size reduction and query latency reduction. The analysis covers four remaining ratios: 25%, 50%, 75%, and 100% (i.e., no pruning). For effectiveness versus efficiency trade-off analysis, we used only the official metrics of the datasets.

Tables 5.6 and 5.7 show the index size and the query processing latency for MS-MARCO and TREC DL 2019 datasets, respectively. Table 5.11 shows the index size and Table 5.15 shows the query processing latency for BEIR dataset. Index size is reported in gigabytes (GB), and query latency is measured in milliseconds (ms). For both metrics, we also include—in parentheses to the left of each value—the factor by which the pruned index size or query latency is reduced compared to the corresponding value for the unpruned PLAID baseline, for each pruning method.

Table 5.6: *Index size (GiB) and query processing latency (ms) on the MS MARCO Passage collection. X denotes the index compression ratio in Index Size and the query processing speed in Latency compared to the PLAID.*

Method	Index size (x)			Latency (x)		
	25%	50%	75%	25%	50%	75%
PLAID		22 (1.0x)			65 (1.0x)	
+ First	5.6 (3.9x)	12 (1.8x)	17 (1.3x)	30 (2.2x)	38 (1.7x)	48 (1.4x)
+ IDF	6.4 (3.4x)	13 (1.7x)	16 (1.4x)	31 (2.1x)	39 (1.7x)	47 (1.4x)
+ TF-IDF	6.3 (3.5x)	13 (1.7x)	16 (1.4x)	30 (2.2x)	39 (1.7x)	48 (1.4x)
+ Attention	5.6 (3.9x)	12 (1.8x)	17 (1.3x)	30 (2.2x)	38 (1.7x)	48 (1.4x)
+ MMTR	6.3 (3.5x)	13 (1.7x)	18 (1.2x)	31 (2.1x)	39 (1.7x)	48 (1.4x)

Table 5.7: *Index size (GiB) and query processing latency (ms) on the TREC DL 2019. X denotes the index compression ratio in Index Size and the query processing speed in Latency compared to the PLAID.*

Method	Index size (x)			Latency (x)		
	25%	50%	75%	25%	50%	75%
PLAID		22 (1.0x)			61 (1.0x)	
+ First	5.6 (3.9x)	12 (1.8x)	17 (1.3x)	31 (2.0x)	37 (1.6x)	47 (1.3x)
+ IDF	6.4 (3.4x)	13 (1.7x)	16 (1.4x)	31 (2.0x)	39 (1.6x)	46 (1.3x)
+ TF-IDF	6.3 (3.5x)	13 (1.7x)	16 (1.4x)	31 (2.0x)	42 (1.5x)	50 (1.2x)
+ Attention	5.6 (3.9x)	12 (1.8x)	17 (1.3x)	30 (2.0x)	38 (1.6x)	48 (1.3x)
+ MMTR	6.3 (3.5x)	13 (1.7x)	18 (1.2x)	31 (2.0x)	39 (1.6x)	49 (1.2x)

Effectiveness versus efficiency trade-off analysis on MS MARCO Passage

Table 5.6, shows the values of compression ratios and factor of query processing time reduction, as well as the PLAID index size and latency. Regarding the index size, we observe that the index sizes of Attention and First tend to occupy slightly less space than the other pruning methods. This observation is consistent with the findings of Lassance et al. [32], who reported similar trends in their study of pruning techniques for ColBERT. We believe that this behavior arises because First and Attention tend to maintain vectors corresponding to punctuation tokens. However, COLBERT and its descendants prune vectors corresponding to punctuation tokens from their indexes [74, 73]. Thus, in practice, the pruning caused by these two methods tends to be slightly greater than that of other methods.

The reported query processing latency required to retrieve the corresponding documents in the end-to-end system was computed as described by Santhanam et al. [73], i.e., we computed the average latency of all queries and then reported the minimum average latency across 3 runs.

With the remaining ratio of 25% the index size is reduced 3.4 to 3.9 times, and the

latency is reduced about 2.1 times. However, this results in a loss of 14.86% in MRR@10 as observed in the last section. Thus, this remaining ratio is shown to be not viable in practice on both in-domain datasets (MS-MARCO and TREC DL 19) due to the severe losses in effectiveness.

The optimal balance between effectiveness and efficiency is achieved with the remaining 50% ratio. Among the methods evaluated, MMTR is the most effective with this ratio, reducing index size and latency by a factor of 1.7, while causing only minor drops of 2.26% in MRR@10 and 1.1% in R@100. Other methods offer slightly better compression but perform worse in terms of effectiveness.

However, it remains unclear which method among MMTR, Attention, or First provides the best overall trade-off at the 50% ratio. Statistically, MMTR is the most effective, outperforming the Attention and First methods by 3.19% and 4.61% in MRR@10 and R@100, respectively. However, its index is 8.33% larger than those produced by the other two methods, and all three methods show similar query latency. TF-IDF and IDF present a slightly less favorable effectiveness-efficiency trade-off compared to MMTR since they share the same index size and query latency as MMTR, but their MRR@10 values are not statistically superior to neither FIRST nor Attentinon. As a result, the choice in this scenario comes down to a trade-off: whether to prioritize effectiveness or minimize storage.

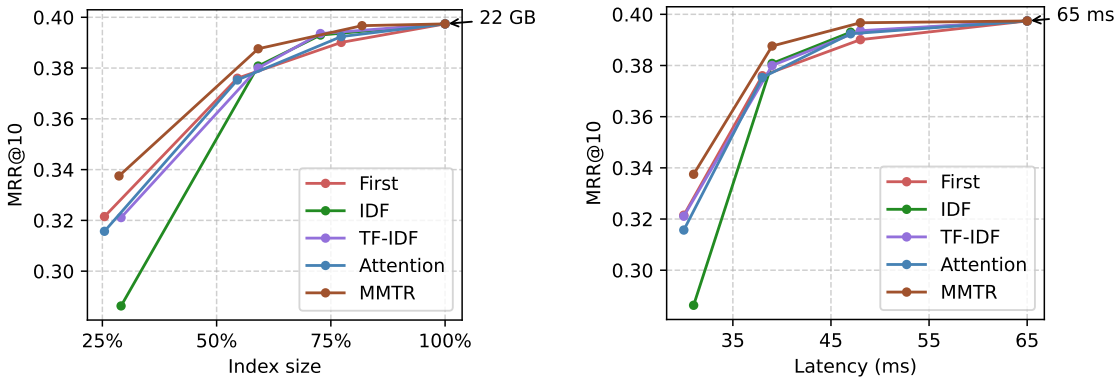
At the remaining 75% ratio, the savings in both index size and query latency are smaller (around 25%) but still meaningful. However, the overall balance between effectiveness and efficiency is noticeably less favorable compared to the 50% scenario. Among the methods, TF-IDF and IDF offers a slightly better trade-offs, achieving moderately better index compression (1.4x) than the others, while matching MMTR and outperforming First and Attention in Recall@100.

Figure 5.1 summarizes the trade-offs between effectiveness and storage usage, as well as effectiveness and latency, across different remaining ratios and pruning methods on the MS MARCO dataset.

Effectiveness versus efficiency trade-off analysis on TREC DL 2019

Recall that TREC DL 2019 uses the same passage corpus as MS MARCO passage, differing only in test set as explained in Subsection 4.1.1. Therefore, we have the same trade-off between effectiveness versus index size as in MS MARCO passage, we refer the reader to Figure 5.1(a).

We show the trade-off between effectiveness versus query processing latency in Figure 5.2, where the nDCG@10 and latency values are taken from Tables 5.1 and 5.7, respectively. We can see more variable latency between the pruning methods at the same remaining ratio (50% and 75%), note that we used a shared machine, these variances can



(a) *MRR@10 versus index size.*

(b) *MRR@10 versus latency.*

Figure 5.1: *Fig. 5.1(a) shows the trade-off between effectiveness (in MRR@10) and memory usage (in GB). Fig. 5.1(b) shows the trade-off between effectiveness (in MRR@10) and query latency (in ms) in the MS-MARCO dataset. The dots in the curves correspond to the remaining ratios (25%, 50%, 75% and 100%).*

be due the processes running in the shared machine from other users or some hardware issues.

The remaining ratio 50% is the best trade-off between effectiveness and query processing latency as illustrated by pruning methods curves. At this setting, MMTR causes losses of 0.40%, while TF-IDF causes a loss of 2.01%. In comparison, other pruning methods causes losses varying from 3.50% to 1.34%.

BEIR Index Size Analysis

Tables 5.8, 5.9, and 5.10 show the index size of each dataset in the BEIR Benchmark, measured in gigabytes across three remaining ratios, note that for each of these tables we show the index size of PLAID without pruning in last column. We aggregate these tables by summing all index sizes across datasets per pruning method and remaining ratio into Table 5.11, which shows how much compression is achieved by pruning methods in general BEIR. In the remaining ratios 25% and 50%, our TF-IDF and MMTR pruning methods can achieve 3.5x and 1.7x storage save compared to PLAID, differing at 75%, TF-IDF achieving greater compression (1.4x) than MMTR, which has 1.2x. In comparison, First, IDF, and Attention have a compression ratio varying from 3.5x to 4.1x for 25% remaining ratio, from 1.7x to 2.1x for 50% remaining ratio, and from 1.3x to 1.4x for 75% remaining ratio.

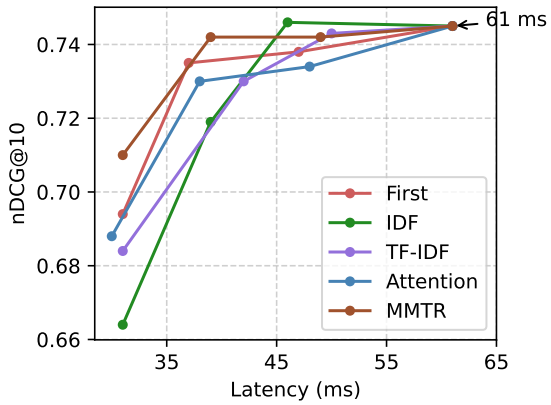


Figure 5.2: *Trade-off between nDCG@10 and latency on the TREC DL 2019. The dots are the remaining ratios (25%, 50%, 75% and 100%). We take MRR@10 and latency numbers from Tables 5.1 and 5.7, respectively.*

Table 5.8: *Index size (in Gigabytes) results for the BEIR Benchmark when we keep 25% of the passage embeddings.*

	TF-IDF	MMTR	IDF	First	Attention	PLAID
DBPedia	3.00	3.00	3.00	2.60	2.50	11.0
FiQA-2018	0.08	0.08	0.08	0.07	0.06	0.27
Natural Questions	2.60	2.60	2.60	2.30	2.20	8.70
HotpotQA	3.10	3.10	3.10	2.70	2.60	11.0
NFCorpus	0.01	0.01	0.01	0.01	0.01	0.03
TREC-COVID	0.30	0.30	0.30	0.26	0.22	1.10
Touché-2020	0.58	0.58	0.59	0.51	0.44	2.00
ArguAna	0.02	0.02	0.02	0.02	0.01	0.06
Climate-Fever	5.20	5.20	5.30	4.50	4.50	18.0
Fever	5.20	5.20	5.30	4.50	4.50	18.0
Quora	0.08	0.08	0.08	0.08	0.07	0.31
SCIDOCS	0.05	0.05	0.05	0.05	0.04	0.17
Scifact	0.01	0.01	0.01	0.01	0.01	0.05

Table 5.9: *Index size (in Gigabytes) results for the BEIR Benchmark when we keep 50% of the passage embeddings.*

	TF-IDF	MMTR	IDF	First	Attention	PLAID
DBPedia	5.90	5.80	5.90	5.20	5.10	11.0
FiQA-2018	0.15	0.16	0.15	0.14	0.13	0.27
Natural Questions	5.10	5.10	5.10	4.50	4.30	8.70
HotpotQA	6.10	6.10	6.10	5.40	5.30	11.0
NFCorpus	0.02	0.02	0.02	0.02	0.02	0.03
TREC-COVID	0.56	0.57	0.57	0.51	0.48	1.10
Touché-2020	1.20	1.20	1.20	1.00	0.96	2.00
ArguAna	0.03	0.03	0.03	0.03	0.03	0.06
Climate-Fever	11.0	11.0	11.0	8.90	8.70	18.0
Fever	11.0	11.0	11.0	8.90	8.70	18.0
Quora	0.17	0.17	0.17	0.17	0.15	0.31
SCIDOCS	0.10	0.10	0.10	0.09	0.08	0.17
Scifact	0.03	0.03	0.03	0.02	0.02	0.05

Table 5.10: *Index size (in Gigabytes) results for the BEIR Benchmark when we keep 75% of the passage embeddings.*

	TF-IDF	MMTR	IDF	First	Attention	PLAID
DBPedia	7.70	8.10	7.70	7.70	7.40	11.0
FiQA-2018	0.18	0.22	0.18	0.21	0.21	0.27
Natural Questions	6.20	7.0	6.20	6.60	6.40	8.70
HotpotQA	7.90	8.40	7.90	7.90	7.70	11.0
NFCorpus	0.02	0.03	0.02	0.03	0.02	0.03
TREC-COVID	0.63	0.82	0.63	0.76	0.75	1.10
Touché-2020	1.30	1.60	1.30	1.50	1.50	2.00
ArguAna	0.04	0.05	0.04	0.04	0.04	0.06
Climate-Fever	13.0	15.0	13.0	14.0	13.0	18.0
Fever	13.0	15.0	13.0	14.0	13.0	18.0
Quora	0.24	0.24	0.24	0.24	0.23	0.31
SCIDOCS	0.11	0.13	0.11	0.13	0.13	0.17
Scifact	0.03	0.04	0.03	0.03	0.03	0.05

Table 5.11: *Index size (GiB) on the BEIR Benchmark. X denotes the index compression ratio compared to the PLAID.*

Method	Index size (x)		
	25%	50%	75%
PLAID		70.68 (1.0x)	
+ First	17.61 (4.0x)	34.87 (2.0x)	53.14 (1.3x)
+ IDF	20.45 (3.5x)	41.37 (1.7x)	50.35 (1.4x)
+ TF-IDF	20.23 (3.5x)	41.36 (1.7x)	50.35 (1.4x)
+ Attention	17.16 (4.1x)	34.39 (2.1x)	50.41 (1.4x)
+ MMTR	20.24 (3.5x)	41.27 (1.7x)	56.62 (1.2x)

BEIR Latency Analysis

We also performed a similar analysis in latency, Tables 5.12, 5.13, and 5.14 show the query processing time of each dataset in the BEIR Benchmark, measured in milliseconds across three remaining ratios. We aggregate these tables by average latency per pruning method and per remaining ratio over the BEIR Benchmark (this is done by summing the query processing time of all 13 datasets and then divide by the number of datasets) into Table 5.15, which shows how much speedup or latency reduction per query is achieved by pruning methods in general BEIR. In the remaining ratios 25% and 50%, our TF-IDF and MMTR pruning methods can achieve 2.1x and 1.5x query processing speedup compared to PLAID, differing at 75%, TF-IDF achieving higher speedup (1.3x) than MMTR, which has 1.2x. In comparison, First, IDF, and Attention have a speedup varying from 2.1x to 2.3x for 25% remaining ratio, from 1.5x to 1.6x for 50% remaining ratio, and from 1.2x to 1.3x for 75% remaining ratio.

Effectiveness versus efficiency trade-off analysis on BEIR Benchmark

As with the in-domain datasets, using a remaining ratio of 25% generally results in significant drops in effectiveness, as discussed in Section 5.1.2. However, this pattern does not apply to all BEIR datasets; in some cases, such as HotpotQA, NFCorpus, Touché-2020, and FEVER, the decrease in effectiveness is minimal. On the other hand, index sizes and query latency are reduced up to 3.6 x and 2.1 x, respectively.

The 25% remaining ratio provides the best trade-off between effectiveness and efficiency for two datasets: TREC-COVID and Climate-Fever, when pruned with MMTR and TF-IDF, respectively. In these cases, index size is reduced by up to 3.6x, and latency is reduced by up to 3.4x, whereas achieving gains in NDCG@10 over PLAID of 3.8% for TREC-COVID and 1.1% for Climate-Fever.

At this remaining ratio, there is no clear winner when considering both effectiveness and storage usage. Attention produces the smallest index but is significantly less

Table 5.12: *Query processing time (in millisecond - ms) results for the BEIR Benchmark when we keep 25% of the passage embeddings.*

	TF-IDF	MMTR	IDF	First	Attention	PLAID
DBPedia	38	33	32	32	32	64
FiQA-2018	23	24	23	22	20	41
Natural Questions	32	33	33	31	32	65
HotpotQA	35	35	35	33	33	69
NFCorpus	17	17	16	17	15	35
TREC-COVID	30	29	30	27	26	61
Touché-2020	35	29	28	32	25	59
ArguAna	32	32	32	30	28	79
Climate-Fever	43	43	42	40	38	122
Fever	35	42	40	40	36	85
Quora	21	21	19	18	18	39
SCIDOCS	20	20	21	20	20	31
Scifact	19	18	19	17	18	24

Table 5.13: *Query processing time (in millisecond - ms) results for the BEIR Benchmark when we keep 50% of the passage embeddings.*

	TF-IDF	MMTR	IDF	First	Attention	PLAID
DBPedia	53	50	45	49	43	64
FiQA-2018	33	35	33	31	30	41
Natural Questions	44	46	42	43	42	65
HotpotQA	52	54	52	49	48	69
NFCorpus	22	23	20	21	18	35
TREC-COVID	42	46	44	40	39	61
Touché-2020	36	36	38	39	32	59
ArguAna	51	51	51	48	46	79
Climate-Fever	59	60	59	64	60	122
Fever	48	54	54	57	54	85
Quora	24	25	23	23	22	39
SCIDOCS	28	28	28	25	24	31
Scifact	23	25	23	21	20	24

Table 5.14: *Query processing time (in millisecond - ms) results for the BEIR Benchmark when we keep 75% of the passage embeddings.*

	TF-IDF	MMTR	IDF	First	Attention	PLAID
DBPedia	64	68	59	64	56	64
FiQA-2018	31	34	31	32	32	41
Natural Questions	56	58	56	57	57	65
HotpotQA	63	67	63	65	64	69
NFCorpus	25	29	26	28	25	35
TREC-COVID	52	53	50	49	46	61
Touché-2020	44	45	45	56	48	59
ArguAna	61	63	60	60	59	79
Climate-Fever	74	79	74	77	75	122
Fever	59	67	59	68	65	85
Quora	28	29	27	26	26	39
SCIDOCS	32	34	32	32	32	31
Scifact	24	26	24	25	24	24

effective than TF-IDF and MMTR. As a result, a trade-off is necessary, and the choice must favor either effectiveness or storage efficiency.

The 50% remaining ratio represents the optimal trade-off between effectiveness and efficiency for most pruning methods in BEIR among the three remaining ratios evaluated. As noted in Section 5.1.2, at this ratio the average loss in effectiveness is only 0.97%, while storage savings range from 1.7x to 2.1x, and query latency improves by approximately 1.5x.

At this ratio, the group of methods $S_1 = \{$ TF-IDF, MMTR, IDF $\}$ consistently outperforms the group $S_2 = \{$First, Attention$\}$ by approximately 10% in NDCG@10. However, as shown in Table 5.11, the methods in group S_1 produce pruned indices that require about 20% more storage (approximately 5 to 6 GB) than those generated by the methods in group S_2, while the query latency is roughly the same across all pruning methods. Therefore, users must choose between higher effectiveness (by selecting a method from group S_1) and greater space efficiency (by choosing a method from group S_2).

At the 75% remaining ratio, the trade-off between effectiveness and efficiency favors effectiveness, with only marginal lossses in all datasets and index size reductions of up to 1.4x and query latency improvements of up to 1.3x[3] In this setting, MMTR still outperforms First and Attention in terms of effectiveness, although its index requires

[3]These values are much smaller than those for the the 50% remaining ratio.

Table 5.15: *Query processing time (in millisecond - ms) on the BEIR Benchmark. X denotes the speedup compared to the PLAID.*

Method	Latency (x)		
	25%	50%	75%
PLAID		60 (1.0x)	
+ First	28 (2.1x)	39 (1.5x)	49 (1.2x)
+ IDF	28 (2.1x)	39 (1.5x)	47 (1.3x)
+ TF-IDF	29 (2.1x)	40 (1.5x)	47 (1.3x)
+ Attention	26 (2.3x)	37 (1.6x)	47 (1.3x)
+ MMTR	29 (2.1x)	41 (1.5x)	50 (1.2x)

more space than Attention's, about 14% more, or roughly 7 GB. TF-IDF and IDF offer a slightly better balance between effectiveness and efficiency: their NDCG@10 scores are only slightly lower than MMTR's, while providing index compression and query latency on par with, or better than, those of First and Attention.

5.1.4 Ablation studies

In this section, we present two ablation studies of the MMTR method. The first study evaluates different pooling strategies and the second investigates the impact of the token grouping problem (i.e., the presence of different vectors corresponding to a same token in the ranking) on the effectiveness of PLAID.

Pooling strategy

We investigated four pooling strategies, named *CLS*, *mean*, *sum*, and *max pooling* on the MS MARCO Passage dataset, using 75% remaining ratio. The CLS pooling uses only the projection of the CLS vector of a document d over a MLM head. The average and sum pooling obtain a unique vector in $\mathbb{R}^{|\text{BERT}|}$ by averaging or summing, respectively, the values for each token i in each of the projections of vectors corresponding to tokens of a document d. The max pooling is explained in Section 3.2.2. As shown in Table 5.16, the max pooling strategy is slightly better than sum and average pooling, whereas CLS is worst one. Thus, we adopted max pooling as the pooling strategy in MMTR.

Token grouping problem

Our proposed MMTR method has two components: MLM Max and reordering algorithm. The first is MLM Max, which consists of ranking tokens through projections generated by MLM head for each token vector as described by Ram et al. [67], using the max-pooling on these projections to get a single rank of tokens as suggested by the

Table 5.16: *Pooling strategy ablation of MMTR method using 75% remaining ratio. Bold denotes the best pooling strategy for that metric.*

Pooling				MS MARCO Passage	
CLS	Mean	Sum	Max	MRR@10	R@100
✓	×	×	×	0.359	0.876
×	✓	×	×	0.393	0.913
×	×	✓	×	0.393	0.913
×	×	×	✓	**0.396**	**0.914**

pooling strategy ablation 5.1.4, and sorting the tokens occurring in the document based on their aggregated scores. The second component is the reordering algorithm that takes as input a list of tokens and returns a list of prioritized tokens, where the first occurrence of tokens is prioritized, and the subsequent occurrences of the same token are recursively pushed down in the new rank as described in Algorithm 3.1, mitigating the token grouping problem.

Table 5.17 shows the impact of the MLM Max versus reordering algorithm (represented as Algorithm 3.1 due to space), or the combination of both components, on the effectiveness of PLAID. In row 1, we remove the reordering algorithm. In row 2, we remove MLM Max and apply the reordering algorithm to the document's tokens, i.e., consider the document d = ['the', 'cat', 'chased', 'the', 'dog', 'and', 'the', 'dog', 'barked', 'at', 'the', 'cat'][4] instead of first assigning scores to tokens and sorting them, the reordering algorithm processes d directly by reordering tokens based on their occurrence patterns, resulting in the output d' = ['the', 'cat', 'chased', 'dog', 'and', 'barked', 'at', 'the', 'dog', 'cat', 'the', 'the'] which prioritizes the first occurrence of each token while placing subsequent repetitions later in the sequence. We find that the reordering algorithm alone slightly outperforms MLM Max alone (row 2 vs 1) in MRR@10 in the MS MARCO Passage at remaining ratios 25%, and 50%. In R@100 in all remaining ratios (25%, 50%, and 75%), the MLM Max alone slightly outperforms reordering algorithm alone (row 1 vs 2). This observation indicates that MLM Max and reordering algorithm improve metrics in different ways and that combining them further improves the retrieval effectiveness (row 3) in the remaining ratios 25%, and 50%. However, in the remaining ratio 75%, MLM Max alone slightly outperforms the reordering algorithm alone and the combination of both components in the MRR@10 and R@100 metrics (row 1 vs 2 and row 1 vs 3). Our default condition is based on these results, the final proposed method uses the reordering algorithm for remaining ratios 25%, and 50%, and disables it for the remaining ratios 75%.

[4] A tokenized version of the document: "the cat chased the dog and the dog barked at the cat."

Table 5.17: *MLM Max versus reordering algorithm ablation. Bold denotes the best result for that metric.*

#	MLM Max	Algorithm 3.1	MS MARCO Passage					
			25%		50%		75%	
			MRR@10	R@100	MRR@10	R@100	MRR@10	R@100
(1)	✓	×	0.310	0.821	0.370	0.900	**0.396**	**0.914**
(2)	×	✓	0.327	0.781	0.385	0.885	0.395	0.910
(3)	✓	✓	**0.338**	**0.847**	**0.388**	**0.905**	0.395	0.912

5.2 Discussion

In this section, we discuss the results of Section 5.1 in the light of our three research questions stated in Section 1.1, as follows in Subsection 5.2.1, 5.2.2, and 5.2.3, respectively.

5.2.1 Differences in the effectiveness of pruning methods on PLAID

In this subsection, we address RQ1 - *What differences exist in the effectiveness of various pruning methods when applied to PLAID output vectors?* – Experiments conducted on the 15 datasets, using recommended experimental protocols from the literature reveal statistically significant differences in effectiveness among the pruning methods evaluated in this study.

MMTR emerged as the most stable method, outperforming or matching TF-IDF and IDF on the majority of datasets, and showing no statistically significant inferiority to any other pruning method across all 15 datasets. TF-IDF ranked next in terms of stability, particularly on out-of-domain datasets. IDF, on the other hand, showed inconsistent performance, making it difficult to draw clear statistical comparisons with other methods in certain scenarios.

Finally, the First and Attention methods consistently demonstrated statistically inferior effectiveness, specifically in Recall@100 on in-domain datasets and NDCG@10 on out-of-domain datasets, when compared to MMTR and TF-IDF in most cases.

5.2.2 Impact of pruning methods on the balance between effectiveness and efficiency for PLAID

Regarding RQ2 - *Do different pruning methods vary in how they affect the trade-off between effectiveness and efficiency when applied to PLAID output vectors?*– Based on the full set of datasets analyzed, our experiments reveal clear and meaningful differences among the pruning methods. At the 25% and 50% remaining ratios MMTR is the most stable pruning method, consistently outperforming or matching TF-IDF and IDF

in effectiveness. First and Attention are the least effective methods, showing statistically significant inferiority to MMTR and TF-IDF across both in-domain and out-of-domain datasets. However, First and Attention consistently produce smaller index sizes compared to MMTR and IDF. Therefore, at these remaining ratios, users must choose between higher effectiveness and greater memory efficiency. Query latency is roughly the same across all methods.

At the 75% remaining ratio, TF-IDF offers the best trade-off between effectiveness and memory usage on the MS MARCO dataset. Along with IDF, it also provides the most balanced performance on the out-of-domain datasets. Query latency remains nearly the same across all methods.

5.2.3 The effect of the remaining ratio in the trade-off between efficiency and effectiveness

Regarding RQ3 - *Is there an approximate optimal remaining ratio* α *for PLAID output vectors that achieves a good balance between effectiveness and efficiency in both in-domain and out-of-domain datasets?* – For the in-domain datasets, the 25% remaining ratio significantly harms effectiveness, with losses of up to 15% in MRR@10 on MS MARCO and 11% in Recall@100 on TREC DL 2019. As a result, the 25% ratio is not a practical option. The 50% remaining ratio clearly offers the best balance between effectiveness and efficiency.

While the 75% remaining ratio does not provide the optimal trade-off, it is still a viable option. It enables memory savings of 1.2x to 1.4x and reduces query latency by about 1.4x, while incurring only minor effectiveness losses on both in-domain datasets.

For out-of-domain datasets, our experiments show that the optimal remaining ratio varies more significantly across datasets, making it difficult to define a general rule of thumb. For example, while a 25% remaining ratio substantially reduces effectiveness in many out-of-domain datasets, it can offer the best effectiveness-efficiency trade-off in some cases—such as TREC-COVID and Climate-Fever.

The differing effects of pruning on in-domain and out-of-domain datasets can be explained by how PLAID, and dense retrieval methods in general, work. These methods learn to adapt the query and passage vectors of a pretrained language model so that the similarity between query token vectors and passage token vectors is maximized when the passage is relevant to the query. This learning occurs during fine-tuning on a specific training set.

As a result, when PLAID is applied to datasets whose distributions differ from that of the training set (i.e., out-of-domain datasets), it often retrieves token vectors from non-relevant passages. In these cases, pruning tends to remove those non-relevant vectors,

effectively acting as a noise-filtering mechanism. When noisy passage token vectors are abundant, pruning can even improve performance beyond that of unpruned PLAID, as observed in some BEIR datasets in our experiments.

The opposite effect tends to occur with in-domain datasets, which usually share a similar distribution with the training data. In this case, the token vectors from relevant passages in the test set are more likely to be similar to the query vectors. Pruning, therefore, is more likely to remove relevant vectors, resulting in a drop in effectiveness, as seen in the MS MARCO Passage and TREC DL 2019 datasets.

In conclusion, while our experiments on out-of-domain datasets generally showed that a 50% remaining ratio offered the best balance between effectiveness and efficiency, this should not be considered a universal rule of thumb. Instead, similar experiments that vary the remaining ratio should be conducted for each out-of-domain dataset to determine the most effective trade-off for that specific case.

CHAPTER 6

Conclusions and Future Work

6.1 Conclusion

In this paper, we proposed two novel pruning methods—TF-IDF and MMTR—designed to remove vectors of less relevant tokens within each document. These methods aim to achieve a better balance between effectiveness and efficiency in multi-vector retrieval models, which typically suffer from large index sizes and significant latency overhead in end-to-end retrieval systems.

Our first method uses TF-IDF scores to determine the relevance of the token, allowing the pruning of the tokens with the lowest values. Our second method, MMTR, utilizes the output of the MLM head for each token vector. We aggregate these outputs via max-pooling and sort the document tokens based on their aggregated scores. Finally, we apply a reordering algorithm that recursively prioritizes the first occurrence of tokens in the document.

We also conducted an experimental comparative study evaluating the proposed methods against three baseline approaches from prior work: IDF, First, and Attention. Following established experimental protocols, we assessed the statistical significance of differences between methods and evaluated their performance in two scenarios: in-domain and out-of-domain datasets.

Experimental results on the MS MARCO Passage and BEIR datasets demonstrate that both MMTR and TF-IDF substantially reduce storage overhead in PLAID. On MS MARCO Passage, MMTR and TF-IDF achieve index compression ratios of 1.2–3.5x and 1.4–3.5x, respectively, while yielding query processing speedups of 1.4–2.1x and 1.4–2.2x. On BEIR, MMTR and TF-IDF achieve compression ratios of 1.2–3.6x and 1.4–3.5x, with corresponding speedups of 1.2–2.1x and 1.3–2.2x. Regarding retrieval effectiveness, MMTR consistently outperforms other pruning methods—including First, Attention, IDF, and TF-IDF—on the in-domain MS MARCO Passage dataset, particularly at lower remaining ratios (25% and 50%), where improvements are statistically significant. MMTR maintains robust and competitive performance as the remaining ratio increases. TF-IDF also exhibits strong results, especially in terms of recall at 100

(R@100). In contrast, on the TREC DL 2019 dataset, no statistically significant differences in nDCG@10 are observed among the pruning methods. Furthermore, our proposed TF-IDF and MMTR pruning methods exhibit strong performance in zero-shot scenarios, outperforming First and Attention in most cases, as confirmed by the Friedman test with significance validated through the Hommel post hoc analysis.

Our findings indicate that the PLAID index on MS MARCO Passage, TREC DL 2019, and BEIR can be pruned by up to 1.7x with minimal loss in retrieval effectiveness. This pruning yields a reduction in end-to-end query latency by a factor of 1.7 on MS MARCO Passage and TREC DL 2019, and by 1.5 on BEIR. In contrast to prior work on static token pruning for multi-vector retrieval models, we followed rigorous experimental protocols recommended in the literature to ensure valid comparisons—both within individual datasets (e.g., MS MARCO and TREC DL 2019) and across multiple datasets (e.g., BEIR benchmark). This methodological rigor enabled us to draw conclusions not previously established. Notably, MMTR demonstrates consistent effectiveness across all remaining ratios, performing well in both in-domain settings (MS MARCO Passage and TREC DL 2019) and zero-shot scenarios (BEIR). TF-IDF also exhibits strong performance in zero-shot settings. In comparison, IDF exhibits unstable effectiveness, particularly at the 25% remaining ratio, making its performance inconclusive. Other approaches, such as First and Attention, do not yield statistically significant improvements and consistently underperform in terms of retrieval effectiveness.

6.2 Future Work

We identified several key limitations of this study, which also suggest promising directions for future research.

First, we focused on pruning the token vectors of the PLAID, experimenting with five pruning strategies, including two novel methods proposed in this study. While our approach demonstrated promising results within the PLAID, we did not extend our analysis to other late interaction models such as ColBERT and ColBERTv2. Future work could explore the generalizability and effectiveness of the proposed pruning techniques across these and other multi-vector retrieval architectures to assess their broader applicability and performance impact.

Second, while the document encoder was fine-tuned for the ranking task, the weights of the MLM head remained unchanged [67]. Additionally, the MLM head was not trained to decode representations of special tokens such as [CLS] or [D]. Given that the MLM head and softmax, as used during BERT pretraining, are optimized to output semantically contextualized token probabilities—not ranking-relevant scores—this misalignment may limit effectiveness. Fine-tuning the MLM head specifically for the ranking

task could better align token scoring with retrieval objectives, potentially improving overall performance.

Third, prior work by Lin et al. [40] has shown that the MLM head, when used without fine-tuning and with softmax normalization, fails to capture the relative importance of tokens. Their findings emphasize that learned token weights often outperform rankings derived solely from MLM and softmax outputs. Future work could explore integrating these learned token weights with the MLM-derived token ranks to produce more reliable token importance estimates, thereby improving pruning decisions and retrieval effectiveness.

Fourth, MMTR currently does not incorporate centroid-based information for pruning, despite its potential utility. In the context of multi-vector retrieval models such as PLAID, token distances from assigned centroids may serve as a valuable relevance signal. We hypothesize that token vectors exhibiting large distances from their assigned centroids contribute minimally to retrieval effectiveness and could therefore be pruned with negligible impact. Conversely, vectors closest to centroids may hold higher relevance for queries. For instance, retaining only the token vector nearest to a given centroid (e.g., C_1) while discarding others mapped to the same centroid may offer an effective pruning strategy.

Bibliography

[1] ACQUAVIA, A.; MACDONALD, C.; TONELLOTTO, N. **Static pruning for multi-representation dense retrieval**. In: *Proceedings of the ACM Symposium on Document Engineering 2023*, DocEng '23, New York, NY, USA, 2023. Association for Computing Machinery.

[2] AKKALYONCU YILMAZ, Z.; WANG, S.; YANG, W.; ZHANG, H.; LIN, J. **Applying BERT to document retrieval with birch**. In: Padó, S.; Huang, R., editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, p. 19–24, Hong Kong, China, Nov. 2019. Association for Computational Linguistics.

[3] BOUALILI, L. **Deep learning for information retrieval : studying relevant signals for ad hoc search based on transformer models**. Theses, Université Paul Sabatier - Toulouse III, Nov. 2022.

[4] BRUCH, S. **Foundations of Vector Retrieval**. Springer Nature Switzerland, 2024.

[5] BURGES, C. J. C. **From RankNet to LambdaRank to LambdaMART: An overview**. Technical report, Microsoft Research, 2010.

[6] CAO, Q.; TRIVEDI, H.; BALASUBRAMANIAN, A.; BALASUBRAMANIAN, N. **DeFormer: Decomposing pre-trained transformers for faster question answering**. In: Jurafsky, D.; Chai, J.; Schluter, N.; Tetreault, J., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, p. 4487–4497, Online, July 2020. Association for Computational Linguistics.

[7] CARMEL, D.; COHEN, D.; FAGIN, R.; FARCHI, E.; HERSCOVICI, M.; MAAREK, Y. S.; SOFFER, A. **Static index pruning for information retrieval systems**. In: *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, p. 43–50, New York, NY, USA, 2001. Association for Computing Machinery.

[8] Craswell, N.; Jones, R.; Dupret, G.; Viegas, E., editors. **Proceedings of the 2009 workshop on Web Search Click Data, WSCD@WSDM 2009, Barcelona, Spain, February 9, 2009**. ACM, 2009.

[9] CRASWELL, N.; MITRA, B.; YILMAZ, E.; CAMPOS, D.; VOORHEES, E. M. **Overview of the TREC 2019 deep learning track**. *Computing Research Repository*, abs/2003.07820, 2020.

[10] DAI, Z.; CALLAN, J. **Context-aware term weighting for first stage passage retrieval**. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, p. 1533–1536, New York, NY, USA, 2020. Association for Computing Machinery.

[11] DEMŠAR, J. **Statistical comparisons of classifiers over multiple data sets**. *J. Mach. Learn. Res.*, 7:1–30, Dec. 2006.

[12] DEVLIN, J.; CHANG, M.-W.; LEE, K.; TOUTANOVA, K. **BERT: Pre-training of deep bidirectional transformers for language understanding**. In: Burstein, J.; Doran, C.; Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, p. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[13] FAN, Y.; XIE, X.; CAI, Y.; CHEN, J.; MA, X.; LI, X.; ZHANG, R.; GUO, J.; LIU, Y. **Pre-training methods in information retrieval**. *Computing Research Repository*, abs/2111.13853, 2021.

[14] FORMAL, T.; LASSANCE, C.; PIWOWARSKI, B.; CLINCHANT, S. **SPLADE v2: Sparse lexical and expansion model for information retrieval**. *CoRR*, abs/2109.10086, 2021.

[15] FORMAL, T.; PIWOWARSKI, B.; CLINCHANT, S. **Splade: Sparse lexical and expansion model for first stage ranking**. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, p. 2288–2292, New York, NY, USA, 2021. Association for Computing Machinery.

[16] FRIAS, J. A. L.; MATOS, S. G. A. D. **Dense and hybrid models for information retrieval**. J. Frias, Aveiro, 2022.

[17] FURNAS, G. W.; LANDAUER, T. K.; GOMEZ, L. M.; DUMAIS, S. T. **The vocabulary problem in human-system communication**. *Commun. ACM*, 30(11):964–971, nov 1987.

[18] GAO, L.; DAI, Z.; CALLAN, J. **Modularized transfomer-based ranking framework**. In: Webber, B.; Cohn, T.; He, Y.; Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, p. 4180–4190, Online, Nov. 2020. Association for Computational Linguistics.

[19] GAO, L.; DAI, Z.; CALLAN, J. **COIL: Revisit exact lexical match in information retrieval with contextualized inverted list**. In: Toutanova, K.; Rumshisky, A.; Zettlemoyer, L.; Hakkani-Tur, D.; Beltagy, I.; Bethard, S.; Cotterell, R.; Chakraborty, T.; Zhou, Y., editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, p. 3030–3042, Online, June 2021. Association for Computational Linguistics.

[20] GAO, L.; DAI, Z.; CHEN, T.; FAN, Z.; VAN DURME, B.; CALLAN, J. **Complement lexical retrieval model with semantic residual embeddings**. In: *Advances in Information Retrieval: 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 – April 1, 2021, Proceedings, Part I*, p. 146–160, Berlin, Heidelberg, 2021. Springer-Verlag.

[21] GARCÍA, S.; FERNÁNDEZ, A.; LUENGO, J.; HERRERA, F. **Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power**. *Information Sciences*, 180(10):2044–2064, 2010. Special Issue on Intelligent Distributed Information Systems.

[22] GUO, J.; CAI, Y.; FAN, Y.; SUN, F.; ZHANG, R.; CHENG, X. **Semantic models for the first-stage retrieval: A comprehensive review**. *ACM Transactions on Information Systems*, 40(4):1–42, Mar. 2022.

[23] HAMBARDE, K. A.; PROENÇA, H. **Information retrieval: Recent advances and beyond**. *IEEE Access*, 11:76581–76604, 2023.

[24] HOFSTÄTTER, S.; ALTHAMMER, S.; SCHRÖDER, M.; SERTKAN, M.; HANBURY, A. **Improving efficient neural ranking models with cross-architecture knowledge distillation**. *Computing Research Repository*, abs/2010.02666, 2020.

[25] HOFSTÄTTER, S.; KHATTAB, O.; ALTHAMMER, S.; SERTKAN, M.; HANBURY, A. **Introducing neural bag of whole-words with colberter: Contextualized late interactions using enhanced reduction**. In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, p. 737–747, New York, NY, USA, 2022. Association for Computing Machinery.

[26] HOFSTÄTTER, S.; LIPANI, A.; ALTHAMMER, S.; ZLABINGER, M.; HANBURY, A. **Mitigating the position bias of transformer models in passage re-ranking**. In: *Advances in Information Retrieval: 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 – April 1, 2021, Proceedings, Part I*, p. 238–253, Berlin, Heidelberg, 2021. Springer-Verlag.

[27] HUMEAU, S.; SHUSTER, K.; LACHAUX, M.; WESTON, J. **Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring**. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[28] JOHNSON, J.; DOUZE, M.; JÉGOU, H. **Billion-scale similarity search with gpus**. *IEEE Transactions on Big Data*, 7(3):535–547, 2021.

[29] KARPUKHIN, V.; OGUZ, B.; MIN, S.; LEWIS, P.; WU, L.; EDUNOV, S.; CHEN, D.; YIH, W.-T. **Dense passage retrieval for open-domain question answering**. In: Webber, B.; Cohn, T.; He, Y.; Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, p. 6769–6781, Online, Nov. 2020. Association for Computational Linguistics.

[30] KHATTAB, O.; ZAHARIA, M. **Colbert: Efficient and effective passage search via contextualized late interaction over bert**. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, p. 39–48, New York, NY, USA, 2020. Association for Computing Machinery.

[31] KWIATKOWSKI, T.; PALOMAKI, J.; REDFIELD, O.; COLLINS, M.; PARIKH, A.; ALBERTI, C.; EPSTEIN, D.; POLOSUKHIN, I.; DEVLIN, J.; LEE, K.; TOUTANOVA, K.; JONES, L.; KELCEY, M.; CHANG, M.-W.; DAI, A. M.; USZKOREIT, J.; LE, Q.; PETROV, S. **Natural questions: A benchmark for question answering research**. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019.

[32] LASSANCE, C.; MAACHOU, M.; PARK, J.; CLINCHANT, S. **A study on token pruning for colbert**. *CoRR*, abs/2112.06540, 2021.

[33] LE, Q. V.; MIKOLOV, T. **Distributed representations of sentences and documents**, 2014.

[34] LEE, J.; DAI, Z.; DUDDU, S. M. K.; LEI, T.; NAIM, I.; CHANG, M.-W.; ZHAO, V. Y. **Rethinking the role of token retrieval in multi-vector retrieval**. In: *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[35] LEE, K.; CHANG, M.-W.; TOUTANOVA, K. **Latent retrieval for weakly supervised open domain question answering**. In: Korhonen, A.; Traum, D.; Màrquez, L.,

editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, p. 6086–6096, Florence, Italy, July 2019. Association for Computational Linguistics.

[36] LI, H. **Learning to Rank for Information Retrieval and Natural Language Processing: Second Edition**. Morgan & Claypool Publishers, 2nd edition, 2014.

[37] LI, M.; LIN, S.-C.; OGUZ, B.; GHOSHAL, A.; LIN, J.; MEHDAD, Y.; YIH, W.-T.; CHEN, X. **CITADEL: Conditional token interaction via dynamic lexical routing for efficient and effective multi-vector retrieval**. In: Rogers, A.; Boyd-Graber, J.; Okazaki, N., editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, p. 11891–11907, Toronto, Canada, July 2023. Association for Computational Linguistics.

[38] LIN, J.; MA, X. **A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques**. *CoRR*, abs/2106.14807, 2021.

[39] LIN, J.; NOGUEIRA, R.; YATES, A. **Pretrained Transformers for Text Ranking: BERT and Beyond**. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2021.

[40] LIN, S.-C.; LI, M.; LIN, J. **Aggretriever: A simple approach to aggregate textual representations for robust dense passage retrieval**. *Transactions of the Association for Computational Linguistics*, 11:436–452, 05 2023.

[41] LIU, Q.; GUO, G.; MAO, J.; DOU, Z.; WEN, J.-R.; JIANG, H.; ZHANG, X.; CAO, Z. **An analysis on matching mechanisms and token pruning for late-interaction models**. *ACM Trans. Inf. Syst.*, 42(5), Apr. 2024.

[42] LIU, T.-Y. **Learning to rank for information retrieval**. *Found. Trends Inf. Retr.*, 3(3):225–331, mar 2009.

[43] LIU, Y.; OTT, M.; GOYAL, N.; DU, J.; JOSHI, M.; CHEN, D.; LEVY, O.; LEWIS, M.; ZETTLEMOYER, L.; STOYANOV, V. **Roberta: A robustly optimized BERT pretraining approach**. *CoRR*, abs/1907.11692, 2019.

[44] LUAN, Y.; EISENSTEIN, J.; TOUTANOVA, K.; COLLINS, M. **Sparse, dense, and attentional representations for text retrieval**, 2021.

[45] LUHN, H. P. **The automatic creation of literature abstracts**. *IBM J. Res. Dev.*, 2(2):159–165, apr 1958.

[46] LUNA, C.; MOYA, A. R.; LUNA, J. M.; VENTURA, S. **Statds library: Statistical tests for data science**. *Neurocomputing*, 595:127877, 2024.

[47] MA, X.; SUN, K.; PRADEEP, R.; LIN, J. **A replication study of dense passage retriever**, 2021.

[48] MACAVANEY, S.; MACDONALD, C.; OUNIS, I. **Streamlining evaluation with ir-measures**. In: Hagen, M.; Verberne, S.; Macdonald, C.; Seifert, C.; Balog, K.; Nørvåg, K.; Setty, V., editors, *Advances in Information Retrieval*, p. 305–310, Cham, 2022. Springer International Publishing.

[49] MACAVANEY, S.; NARDINI, F. M.; PEREGO, R.; TONELLOTTO, N.; GOHARIAN, N.; FRIEDER, O. **Efficient document re-ranking for transformers by precomputing term representations**. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20. ACM, July 2020.

[50] MACAVANEY, S.; YATES, A.; COHAN, A.; GOHARIAN, N. **Cedr: Contextualized embeddings for document ranking**. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '19. ACM, July 2019.

[51] MACAVANEY, S.; YATES, A.; FELDMAN, S.; DOWNEY, D.; COHAN, A.; GOHARIAN, N. **Simplified data wrangling with ir_datasets**. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 2429–2436, 2021.

[52] MARON, M. E.; KUHNS, J. L. **On relevance, probabilistic indexing and information retrieval**. *J. ACM*, 7(3):216–244, jul 1960.

[53] MIUTRA, B.; CRASWELL, N. **An introduction to neural information retrieval**. *Found. Trends Inf. Retr.*, 13(1):1–126, dec 2018.

[54] NASERI, S.; DALTON, J.; YATES, A.; ALLAN, J. **Ceqe: Contextualized embeddings for query expansion**, 2021.

[55] NGUYEN, T.; ROSENBERG, M.; SONG, X.; GAO, J.; TIWARY, S.; MAJUMDER, R.; DENG, L. **MS MARCO: A human generated machine reading comprehension dataset**. In: Besold, T. R.; Bordes, A.; d'Avila Garcez, A. S.; Wayne, G., editors, *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016*, volume 1773 de **CEUR Workshop Proceedings**. CEUR-WS.org, 2016.

[56] NI, J.; HERNANDEZ ABREGO, G.; CONSTANT, N.; MA, J.; HALL, K.; CER, D.; YANG, Y. **Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models**. In: Muresan, S.; Nakov, P.; Villavicencio, A., editors, *Findings of the Association for Computational Linguistics: ACL 2022*, p. 1864–1874, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[57] NI, J.; QU, C.; LU, J.; DAI, Z.; HERNANDEZ ABREGO, G.; MA, J.; ZHAO, V.; LUAN, Y.; HALL, K.; CHANG, M.-W.; YANG, Y. **Large dual encoders are generalizable retrievers**. In: Goldberg, Y.; Kozareva, Z.; Zhang, Y., editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, p. 9844–9855, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics.

[58] NIE, P.; ZHANG, Y.; GENG, X.; RAMAMURTHY, A.; SONG, L.; JIANG, D. **Dc-bert: Decoupling question and document for efficient contextual encoding**. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, p. 1829–1832, New York, NY, USA, 2020. Association for Computing Machinery.

[59] NOGUEIRA, R.; YANG, W.; CHO, K.; LIN, J. **Multi-stage document ranking with bert**, 2019.

[60] NOGUEIRA, R.; YANG, W.; LIN, J.; CHO, K. **Document expansion by query prediction**, 2019.

[61] ONAL, K. D.; ZHANG, Y.; ALTINGOVDE, I. S.; RAHMAN, M. M.; KARAGOZ, P.; BRAYLAN, A.; DANG, B.; CHANG, H.-L.; KIM, H.; MCNAMARA, Q.; ANGERT, A.; BANNER, E.; KHETAN, V.; MCDONNELL, T.; NGUYEN, A. T.; XU, D.; WALLACE, B. C.; RIJKE, M.; LEASE, M. **Neural information retrieval: at the end of the early years**. *Inf. Retr.*, 21(2–3):111–182, jun 2018.

[62] PETERS, M. E.; NEUMANN, M.; IYYER, M.; GARDNER, M.; CLARK, C.; LEE, K.; ZETTLEMOYER, L. **Deep contextualized word representations**. In: Walker, M.; Ji, H.; Stent, A., editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, p. 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[63] QU, Y.; DING, Y.; LIU, J.; LIU, K.; REN, R.; ZHAO, W. X.; DONG, D.; WU, H.; WANG, H. **RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering**. In: Toutanova, K.; Rumshisky, A.; Zettlemoyer, L.; Hakkani-Tur, D.; Beltagy, I.; Bethard, S.; Cotterell, R.; Chakraborty, T.; Zhou, Y.,

editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, p. 5835–5847, Online, June 2021. Association for Computational Linguistics.

[64] RADFORD, A.; NARASIMHAN, K.; SALIMANS, T.; SUTSKEVER, I. **Improving language understanding by generative pre-training**. Technical report, OpenAI, 2018.

[65] RAFFEL, C.; SHAZEER, N.; ROBERTS, A.; LEE, K.; NARANG, S.; MATENA, M.; ZHOU, Y.; LI, W.; LIU, P. J. **Exploring the limits of transfer learning with a unified text-to-text transformer**. *J. Mach. Learn. Res.*, 21(1), Jan. 2020.

[66] RAINIO, O.; TEUHO, J.; KLÉN, R. **Evaluation metrics and statistical tests for machine learning**. *Scientific Reports*, 14(1):6086, 2024.

[67] RAM, O.; BEZALEL, L.; ZICHER, A.; BELINKOV, Y.; BERANT, J.; GLOBERSON, A. **What are you token about? dense retrieval as distributions over the vocabulary**. In: Rogers, A.; Boyd-Graber, J.; Okazaki, N., editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, p. 2481–2498, Toronto, Canada, July 2023. Association for Computational Linguistics.

[68] REIMERS, N.; GUREVYCH, I. **Sentence-BERT: Sentence embeddings using Siamese BERT-networks**. In: Inui, K.; Jiang, J.; Ng, V.; Wan, X., editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, p. 3982–3992, Hong Kong, China, Nov. 2019. Association for Computational Linguistics.

[69] REN, R.; QU, Y.; LIU, J.; ZHAO, W. X.; SHE, Q.; WU, H.; WANG, H.; WEN, J.-R. **RocketQAv2: A joint training method for dense passage retrieval and passage re-ranking**. In: Moens, M.-F.; Huang, X.; Specia, L.; Yih, S. W.-t., editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, p. 2825–2835, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics.

[70] ROBERTSON, S. E.; WALKER, S. **Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval**. In: Croft, B. W.; van Rijsbergen, C. J., editors, *SIGIR '94*, p. 232–241, London, 1994. Springer London.

[71] ROBERTSON, S. E.; WALKER, S.; JONES, S.; HANCOCK-BEAULIEU, M.; GATFORD, M. **Okapi at TREC-3**. In: Harman, D. K., editor, *Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4,*

1994, volume 500-225 de **NIST Special Publication**, p. 109–126. National Institute of Standards and Technology (NIST), 1994.

[72] SALTON, G. **Automatic Information Organization and Retrieval.** McGraw Hill Text, 1968.

[73] SANTHANAM, K.; KHATTAB, O.; POTTS, C.; ZAHARIA, M. **Plaid: An efficient engine for late interaction retrieval**. In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, p. 1747–1756, New York, NY, USA, 2022. Association for Computing Machinery.

[74] SANTHANAM, K.; KHATTAB, O.; SAAD-FALCON, J.; POTTS, C.; ZAHARIA, M. **ColBERTv2: Effective and efficient retrieval via lightweight late interaction**. In: Carpuat, M.; de Marneffe, M.-C.; Meza Ruiz, I. V., editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, p. 3715–3734, Seattle, United States, July 2022. Association for Computational Linguistics.

[75] SHEHATA, D. **Information retrieval with entity linking**, 2024.

[76] SUN, Y.; WANG, S.; LI, Y.; FENG, S.; TIAN, H.; WU, H.; WANG, H. **Ernie 2.0: A continual pre-training framework for language understanding**. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8968–8975, Apr. 2020.

[77] THAKUR, N.; REIMERS, N.; RÜCKLÉ, A.; SRIVASTAVA, A.; GUREVYCH, I. **BEIR: Heterogenous benchmark for zero-shot evaluation of information retrieval models**. In: *Proceedings of the 2021 Neural Information Processing Systems (NeurIPS-2021): Track on Datasets and Benchmarks*, 2021.

[78] TONELLOTTO, N.; MACDONALD, C. **Query embedding pruning for dense retrieval**. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, p. 3453–3457, New York, NY, USA, 2021. Association for Computing Machinery.

[79] URBANO, J.; LIMA, H.; HANJALIC, A. **Statistical significance testing in information retrieval: An empirical analysis of type i, type ii and type iii errors**. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '19, p. 505–514. ACM, July 2019.

[80] VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; POLOSUKHIN, I. **Attention is all you need**, 2023.

[81] WANG, J.; HUANG, J. X.; TU, X.; WANG, J.; HUANG, A. J.; LASKAR, M. T. R.; BHUIYAN, A. **Utilizing bert for information retrieval: Survey, applications, resources, and challenges**. *ACM Comput. Surv.*, 56(7), apr 2024.

[82] WANG, M.; XU, X.; YUE, Q.; WANG, Y. **A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search**. *Proc. VLDB Endow.*, 14(11):1964–1978, jul 2021.

[83] WU, Q.; BURGES, C. J. C.; SVORE, K. M.; GAO, J. **Adapting boosting for information retrieval measures**. *Inf. Retr.*, 13(3):254–270, jun 2010.

[84] WU, Y.; SCHUSTER, M.; CHEN, Z.; LE, Q. V.; NOROUZI, M.; MACHEREY, W.; KRIKUN, M.; CAO, Y.; GAO, Q.; MACHEREY, K.; KLINGNER, J.; SHAH, A.; JOHNSON, M.; LIU, X.; ŁUKASZ KAISER.; GOUWS, S.; KATO, Y.; KUDO, T.; KAZAWA, H.; STEVENS, K.; KURIAN, G.; PATIL, N.; WANG, W.; YOUNG, C.; SMITH, J.; RIESA, J.; RUDNICK, A.; VINYALS, O.; CORRADO, G.; HUGHES, M.; DEAN, J. **Google's neural machine translation system: Bridging the gap between human and machine translation**, 2016.

[85] XIONG, L.; XIONG, C.; LI, Y.; TANG, K.; LIU, J.; BENNETT, P. N.; AHMED, J.; OVERWIJK, A. **Approximate nearest neighbor negative contrastive learning for dense text retrieval**. *CoRR*, abs/2007.00808, 2020.

[86] XU, J.; HE, X.; LI, H. **Deep learning for matching in search and recommendation**. *Foundations and Trends® in Information Retrieval*, 14(2–3):102–288, 2020.

[87] ZHAN, J.; MAO, J.; LIU, Y.; GUO, J.; ZHANG, M.; MA, S. **Optimizing dense retrieval model training with hard negatives**. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, p. 1503–1512, New York, NY, USA, 2021. Association for Computing Machinery.

[88] ZHAN, J.; MAO, J.; LIU, Y.; ZHANG, M.; MA, S. **Repbert: Contextualized text embeddings for first-stage retrieval**. *CoRR*, abs/2006.15498, 2020.

[89] ZHANG, S.; LIANG, Y.; GONG, M.; JIANG, D.; DUAN, N. **Multi-view document representation learning for open-domain dense retrieval**. In: Muresan, S.; Nakov, P.; Villavicencio, A., editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, p. 5990–6000, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[90] ZHAO, W. X.; LIU, J.; REN, R.; WEN, J.-R. **Dense text retrieval based on pretrained language models: A survey**. *ACM Trans. Inf. Syst.*, 42(4), Feb. 2024.

[91] ZIPF, G. K. **Human behavior and the principle of least effort**. Addison-Wesley Press, Oxford, UK, 1949.