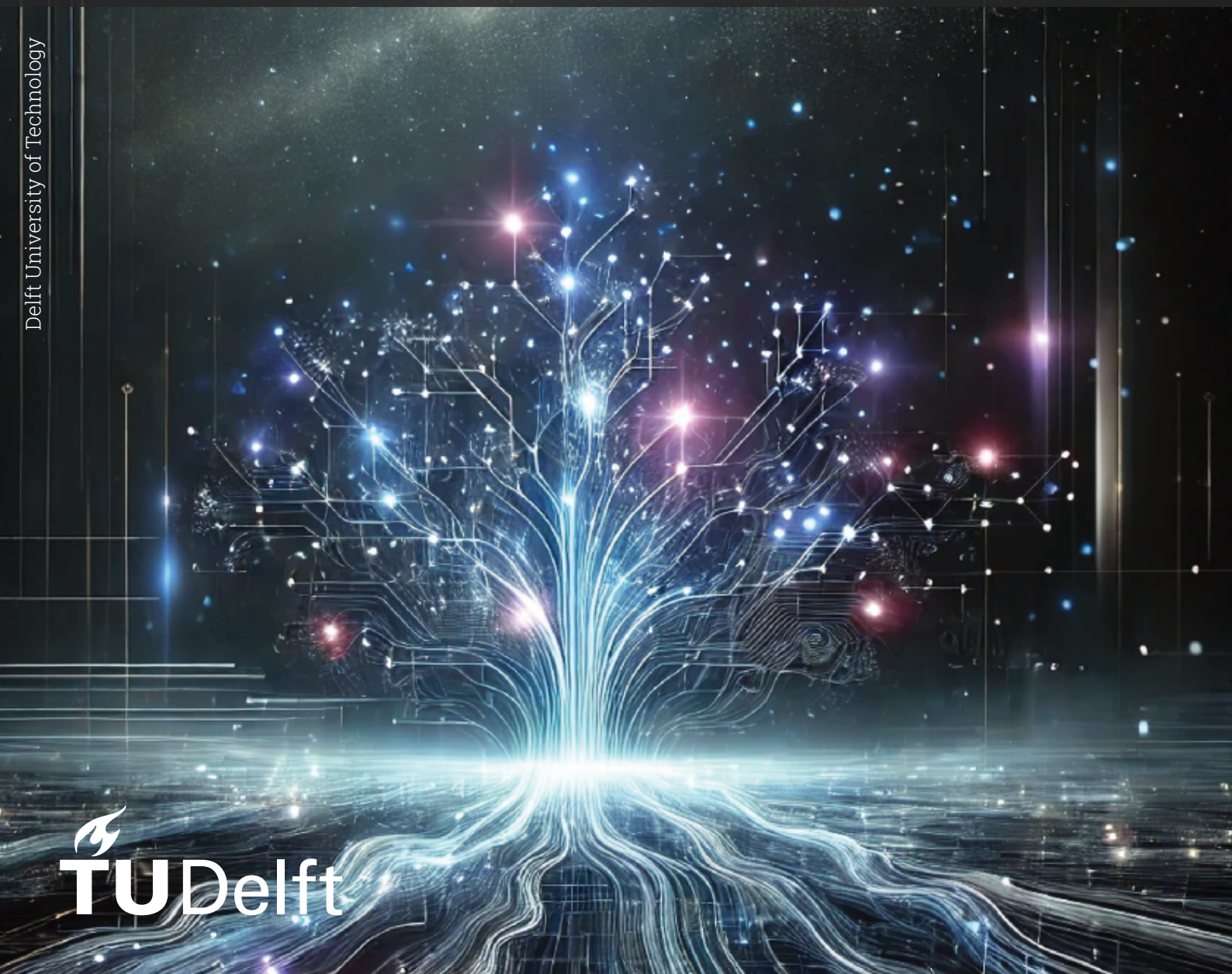
Efficient Query Estimation by Vector Averaging in Dual-Encoder Re-Ranking

Estimating Query Embeddings as Weighted Average of Document Embeddings and Lightweight Query Encoding

Master Thesis (IN5000)

Bo van den Berg



Delft University of Technology

TU Delft

Efficient Query Estimation by Vector Averaging in Dual-Encoder Re-Ranking

Estimating Query Embeddings as Weighted Average of Document Embeddings and Lightweight Query Encoding

by

Bo van den Berg

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Monday March 11, 2025 at 10:00 AM.

Information:

Project duration:	February 13, 2024 – March 11, 2025
Program:	Master Computer Science
Track:	Software Technology
Special program:	Information Architecture
Student number:	4534867

Thesis Supervision:

Thesis Chair, Supervisor:	Prof. Dr. Avishek Anand,	TU Delft
Daily co-supervisor:	Dr. Jurek Leonhardt,	TU Delft
Committee Member:	Dr. Julián Urbano,	TU Delft

Initial credits:

Cover:	Generated on https://chatgpt.com/
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at http://repository.tudelft.nl/.
The source code is available at https://github.com/BovdBerg/fast-forward-indexes [6].

TU Delft

Abstract

A central problem in information retrieval (IR) is passage ranking, where the task is to retrieve passages from a corpus and order them in decreasing relevance to an arbitrary search query. Traditional lexical retrieval methods are susceptible to the vocabulary mismatch problem, where relevant passages are overlooked if they do not contain the exact query terms (e.g., synonyms), despite being semantically relevant. A recent trend in IR is to address this issue by utilizing neural network models (dense rankers) which embed text sequences into dense vector representations that effectively capture their semantics through complex attention mechanisms. For efficiency, dense rankers are often employed in a retrieve-and-re-rank setting, where a lexical ranker initially retrieves a subset of candidate passages, which are then reordered more accurately by a dense ranker.

In this thesis, we focus on the task of passage re-ranking. We employ a dual-encoder architecture as re-ranker that employs a two independent query and document encoders, allowing document embeddings to be pre-computed. Dense query-passage similarity is computed as a dot product between their representations. We then combine scores from both stages using score interpolation.

We identify query encoding latency as a bottleneck and propose an Average Embedding (AvgEmb) estimator. This novel model can efficiently predict an accurate query representation, without requiring any attention-based encoding. It operates solely on looking up embeddings and computing their weighted average representation. Our model is distilled from a TCT-ColBERT and achieves 98.6% of its teacher's accuracy while being 13.4X more efficient in query latency and 1.6X better in the full interpolated passage re-ranking pipeline on CPU.

Our code is publicly available [6].

Preface

When I began my Master's in Computer Science, I was blown away by the vast amount of elective courses available. Narrowing it down to only a few courses felt like such an overwhelming task. I decided to first choose a research group and master track, and choose the recommended courses accordingly. The Web Information Systems group immediately stood out to me, with its focus on the connections between data, information, and humans. This aligned perfectly with my interests in efficient software and Information Architecture. These courses prepared me really well for this thesis. Additionally, trying to keep up with the rapid growth of AI models, I wanted to form a deep understanding of how to train, utilize, and control such models and their capabilities. I hadn't explored AI in such depth before, but I willingly embraced this extra challenge for my final study project.

This past year has been the most challenging period of my academic life. I have felt completely lost, struggling with the concepts of machine learning to the point where I lost almost all motivation to continue with my studies. These thoughts shocked me so much that I reached out to anyone that could help: my supervisors, academic counselor, and even a therapist. These conversations turned everything around. My parents gave me the excellent mental support I needed at the time. Jurek patiently revisited the basics with me, and together with him and Avishek, we decided to pivot to an alternative, novel approach. This new approach instead focuses on estimating the query vector based on its relevant vectors, removing the need for regular query encoding. My enthusiasm, motivation, and confidence quickly returned after this shift, and only grew increasingly stronger. Quickly after, I trained and completely comprehended my first AI model. This realization of my newfound potential felt incredible, and rekindled my passion for the topic. What I had struggled so much for six months ago, I was now able to do within three days. Good results led to better discussions, which brought even greater results. I am finally confident in my work, and even excited to present this research to you.

I am deeply grateful to my supervisors. Your excellent help has been the very foundation of my thesis. I overheard plenty peers complain about unsupportive supervisors, but my experience has been the opposite. Jurek Leonhardt: your unwavering support, personal guidance, and the weekly meetings with insightful discussions and research ideas have always made me feel motivated and inspired. You showed an actual interest in what and how I was doing. Thank you for being so understanding. You have been the best mentor I could have ever wished for. Avishek Anand: I am really glad I chose to conclude my study with you as supervisor. Your understanding and encouragement during my struggles were invaluable. You have been a great motivator for my thesis, always providing great new insights. The follow-up message after hearing about my progress after pivoting showed me you also really cared.

My success would also not have been possible without the amazing support of my family, friends, and supervisors. Your belief in me during moments of self-doubt kept me going. This project has been the most challenging, but also definitely the most rewarding I have ever completed. Looking back, I am incredibly proud of my growth, perseverance, and resilience. This thesis is not only an embodiment of my academic efforts, but also a testament to the resilience and determination I have developed. It showed me how rewarding it can be to push through emotional distress. Thank you to everyone who has been part of this journey — I could not have done it without you.

Bo van den Berg
Delft, March 2025

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
AI	Artificial intelligence
BeIR	Benchmarking IR dataset [80]
DPR	Dense passage retrieval
FF	Fast-Forward indices [45]
FiQA	Financial Question Answering dataset [80]
KD	Knowledge distillaton
MSM	MSMARCO dataset [62]
MSM-Psg	MSMARCO dataset for passage retrieval
NQ	Natural Questions dataset [40]
QA	Question Answering
RNN	Recurrent neural network
SOTA	State of the art
ST	Small transformer
TAS	Topic-aware sampling
Dev	Development

Notations

Symbol	Definition
D	Document set, corpus
d	Document, or passage (equivalent in this context)
Q	Query set
q	Query

Model size:

H	Amount of hidden dimensions
L	Amount of encoder layers

Evaluation metrics:

AP	Average precision
MAP	Mean average precision
MRR	Mean reciprocal rank
nDCG	Normalized discounted cumulative gain
P	Precision
R	Recall
RR	Reciprocal rank

Sampling:

d^-	Negative sample

Symbol	Definition
d^+	Positive sample
Scoring:	
ϕ	Score
ϕ_D	Dense score
ϕ_S	Sparse score

Synonyms

Synonyms	Explanation
{document, passage, doc}	BERT-like models often require a maximum length of input tokens, and thus require larger documents to be split into multiple passages. Within the scope of this thesis, we assume that all input is processed atomically and we do not distinguish between documents and passages.
{corpus, document set, passage set}	Using the reasoning of the [document, passage] synonyms above, the sets are also used interchangeably.
{encoding, embedding (emb), representation (rep), vector}	The vectors created by encoder models to numerically represent the meaning of a document or a query.
{sparse retrieval, lexical retrieval}	Retrieval based on exact term matching, e.g. BM25
{retrieval set, sparse rankings}	Top-k ranked documents from sparse retrieval, e.g., results from retrieval stage in retrieve-and-re-rank setting
{dense retrieval, semantic retrieval, dense passage retrieval (DPR)}	Retrieval using neural ranking models where documents and queries are used as inputs
{encoder, model}	Artificial Intelligence model used to encode queries or documents in the context of neural ranking
{dual-encoder (model), bi-encoder (model)}	Retrieval using neural ranking models where documents and queries are encoded into a vector space and retrieval is done based on (approximate) nearest neighbor search
{cross-attention encoder (model), cross-encoder}	Retrieval using neural ranking models where documents and queries are used as inputs and a similarity score comes out
{inference, online}	Runtime between user entering a search query and returning the list of ranked documents.

1

Introduction

Information retrieval (IR) is the field dedicated to retrieving data from large collections [58]. We consider the domain of ad-hoc information retrieval, where the predominant task is to retrieve a list of k documents from a corpus ordered by their relevance to an input query q [58]. For example, web search engines such as Google perform this task by presenting the most relevant websites and a highlighted passage. Low inference latency (efficiency) and accurate results (performance) are paramount in this scenario, as users expect prompt responses that satisfy their information needs.

Traditional lexical retrieval methods [78, 72] rely on exact matching between terms (e.g. words) in queries and documents. However, they are susceptible to the vocabulary mismatch problem [21] in which documents are not retrieved if they do not contain the exact query terms (e.g. synonyms), even though they are semantically relevant.

In recent years, the use of neural networks has become increasingly prevalent in information retrieval [46]. These dense rankers leverage complex attention mechanisms [81] to capture the meaning (semantics, context) of text sequences into dense vectors [15, 60], leading to highly accurate results in natural language processing tasks such as ranking [46]. The primary limitation of these models is their quadratic time complexity in respect to the input length [51]. In response, the input is often shortened by splitting all documents into shorter passages [46]. Throughout this report, we use the terms passage and documents interchangeably. However, this approach substantially increase the corpus size, and inference latency remains impractical when computing similarities for all query-passage pairs. Many approaches circumvent this problem by employing a *retrieve-and-re-rank* telescoping setting, where an initial lexical retrieval step efficiently identifies a subset of candidate documents, which are then reordered more accurately in a second stage using advanced ranking methods.

One efficient approach to dense ranking is dense passage retrieval (DPR, subsection 2.4.2) [36], which employs a dual-encoder architecture consisting of two independent models that respectively embed the queries and documents into a common vector space and compute their relevance based on similarity metrics between those vector representations. These models are particularly efficient because the documents are known in advance and their representations can be pre-computed and indexed. Recent research by Leonhardt et al. [45] show that can also be employed as efficient and accurate re-rankers. They also show that the lexical and dense scores reveal complementary information [8] and linear score interpolation (Equation 2.2) further boosts the performance at negligible latency cost.

Many applications do not reach sufficient traffic to encode queries in batches, disallowing parallelization hardware such as GPUs/TPUs and necessitating to encoding on a CPU, which increases latency overhead [9]. Additionally, specialized parallelization hardware generally comes with monetary costs

[45] and negative environmental impacts [75].

Considering all of the above, our research is thus aimed at improving inference efficiency in the passage retrieval task on CPU without compromising accuracy. To achieve this, we consider an interpolated retrieve-and-re-rank approach and utilize dual-encoders as re-rankers. We compute the dense similarity score as a dot product between the query vector and document vector (Equation 2.8).

During re-ranking inference, our setup involves these components: creating query vectors, retrieving document vectors from an in-memory index, computing query-document relevancy scores using dot products and score interpolation, and rearranging the candidate documents. Many dual-encoders are homogeneous, employing the same architecture for both encoders [45]. Figure 1.1 illustrates the distribution of re-ranking latency when employing TCT-ColBERT as re-ranker on CPU.

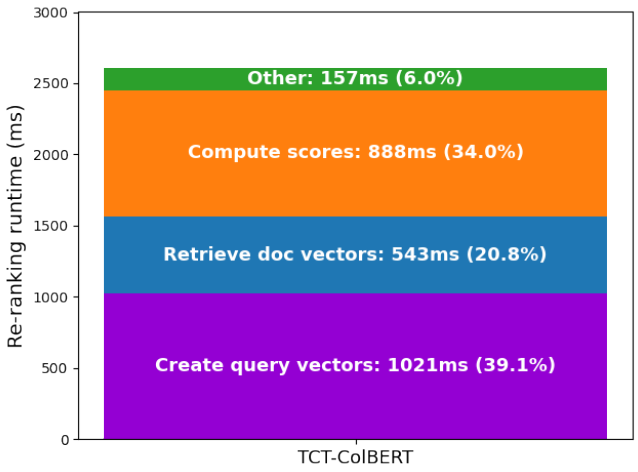


Figure 1.1: Breakdown of homogeneous TCT-ColBERT dual-encoder re-ranking runtime; measured with 10 CPU cores, 128 sampled queries from the MS MARCO [62] development set, retrieval depth $k_S = 1000$, encoder batch size 32, and an in-memory document embedding index [45].

This figure identifies query latency as a bottleneck, making up 39.1% of the runtime. Based on this observation, we shift our research focus towards efficiently finding an accurate query representation. Bruch, Gai, and Ingber [8] reveal that queries are often shorter and more concise than documents. Consequently, Leonhardt et al. [45] propose a lightweight embedding-based query encoder without any complex attention mechanisms [81] (Figure 2.7).

This thesis is driven by the following research questions:

RQ1. Is it possible to achieve an efficient and accurate estimation of a query embedding in neural re-ranking by leveraging the lexically most relevant document embeddings?

RQ2. How can this approach be extended with semantic query information to improve performance without significantly compromising efficiency?

RQ3. What alternative approaches and settings could further boost the model performance or efficiency?

In our research, we extend on the embedding-based query encoder by integrating learned token weights and additionally leveraging the query's n lexically most relevant documents, their embeddings, and weighted average computations. The intuition behind the token weights is to assign higher weights to context-defining tokens (e.g. "why", "capital") and lower weights prevalent and general words (e.g. stop words such as "the").

Central to this thesis, we propose the Average Embedding (AvgEmb) query estimator (Figure 3.4, Equation 3.4). This novel method relies solely on efficient operations: retrieving token embeddings from a

matrix, retrieving document embeddings from an in-memory index, and transforming them into a 1-dimensional vector based on their weighted average representation. Employing this query estimator into our dual-encoder architecture results in a re-ranking pipeline consisting of only efficient operations (Equation 3.5).

Our model comes in two variants, $\text{AvgEmb}_{\text{n-docs}}$ that operates only on the document embeddings, and $\text{AvgEmb}_{\text{q,n-docs}}$ that also incorporates lightweight query encoding. We argue that the first variant is vulnerable to the vocabulary mismatch problem [21] and our experiments confirm the latter model to be strictly superior. Continuing, we explore various model settings and conclude that $n = 10$ is the most effective variant. Documents at higher ranks seem to incur noise, making the final query estimation prediction less accurate. We therefore adopt the $\text{AvgEmb}_{\text{q,10-docs}}$ variant as our default and refer to it as just AvgEmb.

Our model is trained as a distilled version of TCT-ColBERT [48] using MSE loss [82] (Figure 3.7). In this training setup, the teacher's performance serves as an upper boundary on the student's performance. We demonstrate that our proposed estimator achieves 98.6% of its teacher while being 13.4X more efficient in query latency and 1.6X more efficient in the full interpolated passage re-ranking pipeline on CPU. We deduce that query latency could be reduced by 40% if document vectors are re-used throughout the re-ranking pipeline. Reflecting on our original problem: our efficient query estimator significantly enhances ranking efficiency without compromising accuracy, leading to a more satisfactory information retrieval overall.

The remainder of this report is structured as follows. Starting off, chapter 2 summarizes the background information and related work required to fully comprehend our research, while iteratively deducing our research focus. Building forth on this knowledge, chapter 3 then formally illustrates the reasoning and methodology behind our proposed approach. Moving on from theory to practicality, chapter 4 describes the experimental specifics including the data, metrics, baselines, and details about evaluation, training, and implementation. We then present our experiments and results in chapter 5. Finally, chapter 6 presents our discussions, conclusions, limitations, and provides directions for future work.

For reproducibility, transparency, and validity, we made our code publicly available [6].

2

Background and Related Work

This chapter introduces the task of passage re-ranking and provides an overview of the current state of neural ranking models. First, the families of sparse, dense, and hybrid retrieval methods are introduced. The baseline methods for our research are explained in further detail. The final section introduces the required concepts of AI training, specifically knowledge distillation.

2.1. Passage Re-Ranking

In the domain of Ad-Hoc Information Retrieval (IR), the task of document retrieval involves ranking a set of documents based on their relevance to an arbitrary search query [46]. Given an input query q, the task is to retrieve a list of k documents from a corpus C, ordered by their relevance to q. For instance, when entering a query into Google, it aims to return the most relevant information and websites. The queries are unknown in advance, and the task needs to be executed with high efficiency (low latency) and high performance (accuracy). To address computational constraints of neural ranking models [51] and reduce contextual noise, documents are often split into passages, which are ranked individually in a task called passage ranking. A popular choice is to then compute the document scores as the maximum score of its passages by the $maxP$ approach [13]:

$$\phi(q, d) = \max_{p_i \in d} \phi(q, p_i), \tag{2.1}$$

where $\phi_D(q, X)$ denotes the relevancy score between query q and $X \in$ {document, passage}.

Retrieve-and-Re-Rank

This thesis is focused on inference efficiency in passage ranking. In ranking, accuracy and efficiency are generally a trade-off. It is generally infeasible to apply highly accurate ranking models to an entire corpus. Therefore, a popular efficient approach is retrieve-and-re-ranking [46, 45], which involves two stages: lexical retrieval and semantic re-ranking. First-stage retrieval is the act of retrieving a small set of high-recall candidate passages from a corpus, often performed by lexical sparse retrievers for efficiency. These passages are then reordered in the re-ranking phase, generally by more expensive dense models for high accuracy. These families of models are described in section 2.2 and section 2.4, respectively. This pipeline is visualized in Figure 2.1. This research focuses on the re-ranking phase. Nguyen et al. [62] describes the task of passage re-ranking as "Given a candidate top 1000 passages as retrieved by BM25 [72], re-rank the passages by relevancy." In this case, the sparse retrieval depth $k_S = 1000$.

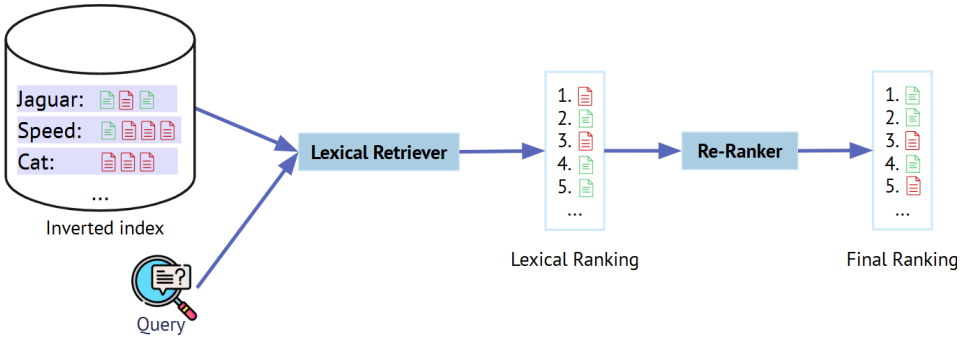


Figure 2.1: Retrieve and re-rank pipeline. Green means relevant, red non-relevant.

Score Interpolation

Sparse and dense scores reveal complementary information from the passages [8], and combining them has been shown to boost re-ranking performance while barely compromising efficiency [45]. A common classical approach is *score interpolation* [83]:

$$\phi(q, d) = \alpha \cdot \phi_S(q, d) + (1 - \alpha) \cdot \phi_D(q, d), \tag{2.2}$$

where ϕ is the final score; ϕ_S is the sparse score; ϕ_D is the dense score; and α is a hyperparameter that decides the impact of either component; q and d represent a query and document, respectively.

The sparse scores ϕ_S are already computed before re-ranking and can be re-used without additional costs. Our research is therefore focused on efficiently computing the dense scores ϕ_D, which can be computationally expensive as complex neural models are applied to 1000 passages for each query.

2.2. Sparse Ranking Models

The traditional ranking approach is sparse retrieval, in which the relevancy score is computed based on exact term matching between query-document pairs, counting the overlapping terms and their frequencies [58]. These lexical methods rely on inverted indexes that store information for each document term. These indexes are pre-computed offline and loaded into random access memory (RAM) before inference. The efficiency of online sparse retrieval stems from its sole reliance on index look-up operations and simple mathematical computations. This concept is fundamental to our research as well.

These methods are referred to as sparse methods because they operate on a sparse matrix of query and document terms, which predominantly contains zeros. Many legacy search engines were powered by sparse retrieval [46], making them sensitive to search engine optimization techniques such as keyword stuffing [91] and hidden texts [27].

Many lexical sparse retrievers are bag-of-words (BoW) models [29], which disregard term order and term context. BoW models and other lexical retrievers [43] lack accuracy because they suffer from the *vocabulary mismatch problem* [21, 71, 46]; that is: semantically relevant documents are not retrieved when they do not contain exact terms from the search query. For instance, document that contain synonyms or related information rather than the exact terms are ignored. This problem forms one of the central challenges in lexical IR [46]. Neural models are generally successful in address this problem by contextualizing the context in which the terms appear. Families of such models include learned sparse retrieval models such as SPLADE [18] and dense rankers [15, 37], both of which are presented in the following sections.

TF-IDF

A foundational sparse ranker is the Term Frequency-Inverse Document Frequency (TF-IDF) method [78], which is slightly adjusted for sparse retrieval here:

$$\phi_S^{TF-IDF}(q,d) = \sum_{t \in q} tf_{t,d} \times log\left(\frac{|C|}{df_t}\right), \tag{2.3}$$

where $tf_{t,d}$ describes the amount of times term t appears in document d (term frequency); $|C|$ is the corpus size; and df_t is the number of documents in which term t appears (document frequency).

Both of these term attributes $tf_{t,d}$ and df_t are pre-computed for all terms in the corpus and stored as inverted indexes. The TD-IDF model is designed to retrieve passages in which the query terms occur often, adjusted for the frequency in which those terms occur overall. For instance, stop words such as "the" are not likely to reveal important information about the query so its impact is reduced by IDF.

BM25

The most popular variation on TF-IDF is Okapi Best Matching 25 (BM25) [72, 71]:

$$\phi_S^{BM25}(q,d) = \sum_{t \in q} \frac{tf_{t,d} \cdot (k_1+1)}{tf_{t,d} + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avgdl}\right)} \cdot log\left(\frac{|C| - df_t + 0.5}{df_t + 0.5}\right), \tag{2.4}$$

where most terms are repeated from the TF-IDF model above; $|d|$ is the length of document d; $avgdl$ is the average document length in the corpus; and k_1 and b are hyperparameters controlling term saturation and the impact of document length on term saturation, respectively.

Due to its effectiveness and simplicity, BM25 remains a widely used scoring function for sparse retrieval. It is an integral part of the MS MARCO passage re-ranking task definition [62]. Most IR research considers BM25 as a baseline, and so do we.

SPLADE

The SParse Lexical AnD Expansion model (SPLADE) [18] is a learned approach for sparse ranking. It enhances sparse document representations using a transformer model and explicit sparsity regularization on term weights, resulting in highly sparse contextually-expanded representations [5, 53]. Additionally, the model employs a log-saturation effect on term weights that prevents terms from becoming overly dominant in the representation.

SPLADE inherits the desirable properties from BoW models [29], including their interpretability and compatibility with inverted indexes. The learned representations mitigate the vocabulary mismatch problem [21] and enable SPLADE to substantially outperform BM25, although this performance boost comes with significantly increased latency for both indexing and online query processing. However, dense dual-encoder re-rankers reach similar performance with reduced overall latency [46]. These are introduced in the next section.

The research on SPLADE has since been extended with an efficiency study [41] and various newer models [20, 19, 42]. The SPLADE++ model [19] serves as one of our baselines and is described in more detail in subsection 2.5.1.

2.3. Transformer Models and Attention

Transformer models were introduced in a foundational paper by Vaswani et al. [81]. The complete architecture of Transformers is very complex, but I made an effort to illustrate it completely in Figure 2.2.

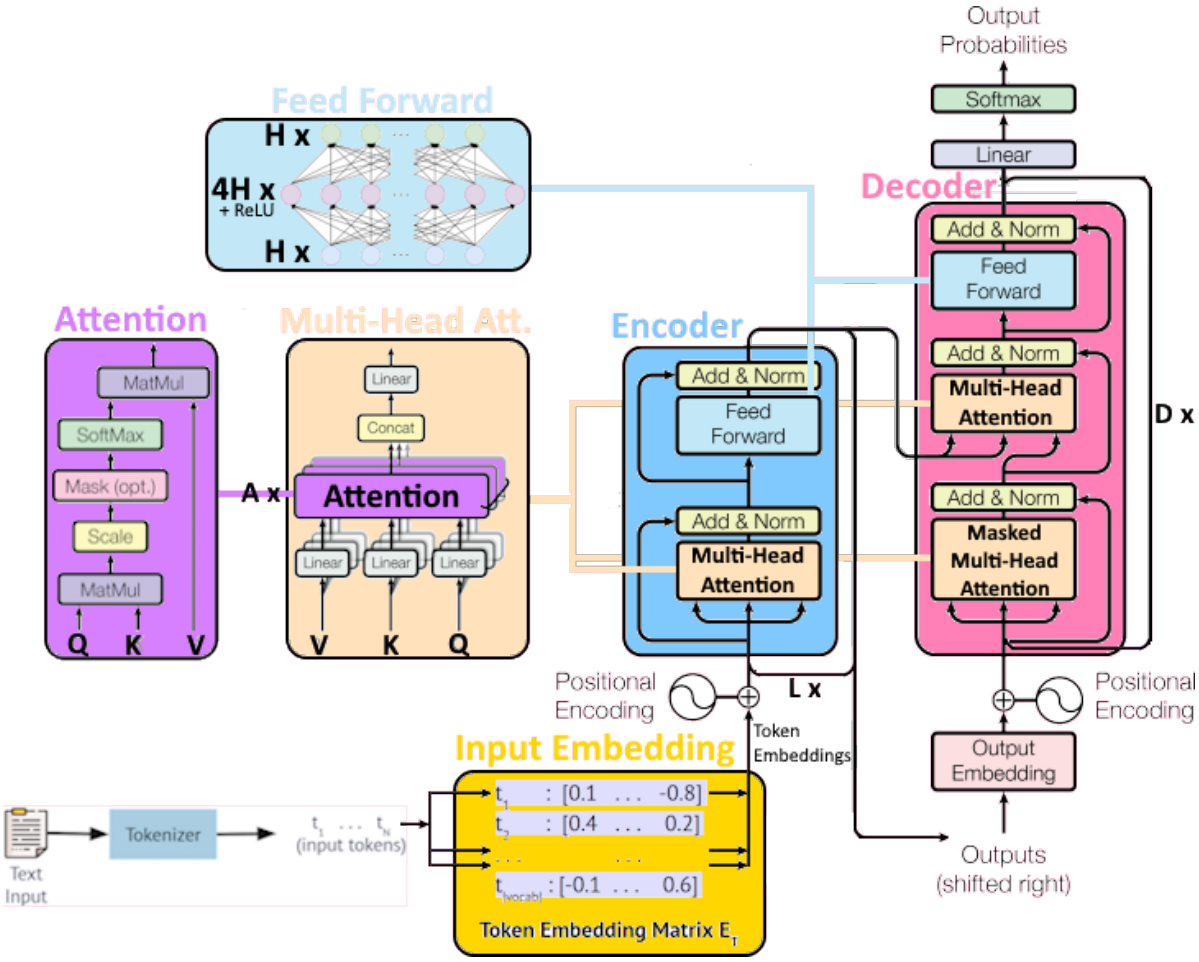


Figure 2.2: Transformer architecture (using attention). Partly copied from Vaswani et al. [81], but adjusted and extended to increase interpretability.

Note that a Transformer model consists of arbitrary amounts of L encoder layers and D decoder layers, which mainly influence their computational performance-efficiency tradeoff. Especially the decoder is not relevant, since BERT-based models [15] (subsection 2.4.1) do not employ any decoder layers. For our contributions, it suffices to understand the embeddings layer and that attention mechanisms are inherently complex and computationally demanding (slow).

Starting from the bottom left, a textual input sequence (e.g. "What is the capital of France") is provided to the Transformer model. This sequence is then tokenized into tokens $t_1 \cdots t_N$. In the input embedding layer, these tokens are transformed to token embeddings, by retrieving their representations from a token embedding matrix E_T following the mapping $E : \mathbb{N} \mapsto \mathbb{R}^H$, such that $E(t)$ is the embedding vector e_t of token t.

For the sake of thoroughness, we still describe the complicated attention mechanism [81, 2] and transformer encoding layers here. Attention-based dense encoders such as BERT-based [15] (i.e. most) models use Transformer encoder layers [81] to transform the embedding vectors e_t into contextual output vectors. BERT models consider the context bidirectionally [15], while GPT models [7] such as the

popular ChatGPT[1] only consider previous sequence tokens.

Each encoder layer has two main components: multi-head attention and a feed-forward sub-layer. Attention accepts three input matrices as inputs: queries Q, keys K, and values V. Since Transformer encoders compute self-attention, Q, K, and V are projections from the output of the previous encoder layer. Attention is computed as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)V. \tag{2.5}$$

Multi-head attention computes attention for each A attention head h_i and concatenates the results:

$$\begin{aligned} MultiHead(Q, K, V) &= (h_1 \circ \cdots \circ h_A) \cdot W^O, \\ h_i &= Attention(QW_i^Q, KW_i^K, VW_i^V), \end{aligned} \tag{2.6}$$

where matrices $W_i^Q \in \mathbb{R}^{H \times d_k}$, $W_i^K \in \mathbb{R}^{H \times d_k}$, $W_i^V \in \mathbb{R}^{H \times d_v}$, $W^O \in \mathbb{R}^{AD_v \times H}$ are trainable parameters; H denotes the dimension of hidden representations in the model; and $d_k = \frac{H}{A}$ is a scaling factor.

2.4. Dense Ranking Models

Dense neural ranking models [15, 37, 48] address the vocabulary mismatch problem by leveraging neural networks to capture the contextual meaning of queries and documents, facilitating semantic matching through representation learning [46]. All terms in a query, document, or query-document pair are transformed into low-dimensional dense vector representations by the model using self-attention mechanisms [81, 15, 2]. section 2.3 illustrates the details and computational complexity behind these attention mechanisms. Dense vector means that most dimensions are represented with non-zero values. The input is often prepended by a classification token [CLS] that aggregates information about the entire input sequence into a single dense vector [15]. Two common architectures of dense ranking models are cross-encoders and dual-encoders, introduced in subsection 2.4.1 and subsection 2.4.2, respectively.

Dense ranking models can be used directly for dense passage retrieval (DPR) [36] or in the re-ranking setting. Neural ranking typically employs large BERT-based transformer models [15, 37] making it computationally expensive and significantly slower than traditional sparse retrievers such as BM25 [72], but also highly accurate. Dense retrieval models generally have lower recall than term-matching sparse models at higher retrieval depths [45]. One approach to reduce computational complexity is knowledge distillation, described in subsection 2.5.1.

One significant limitation of transformer-based architectures for dense encoders is their quadratic time complexity concerning input length, restricting the number of input tokens [46]. Strategies to address this limitation include document truncation [52] and chunking documents into passages [73, 13]. However, chunking significantly increases the corpus size, leading to varied negative latency concerns for the different types of dense ranking models. Training and inference on large corpora are infeasible for many due to computational costs, often necessitating expensive parallelization hardware like GPUs/TPUs, accompanied by high emissions that negatively impact the environment [75].

[1] ChatGPT: https://chatgpt.com

2.4.1. Cross-Encoders

One approach to dense ranking exhibits cross-attention models, which take a concatenated query-document pair as input and compute their relevance score $\phi_D(q, d)$ directly into the classification token via cross-attention, as illustrated in Figure 2.3. An example input is "$[CLS]\ q_1 \cdots q_N\ [SEP]\ d_1 \cdots d_M\ [SEP]$".

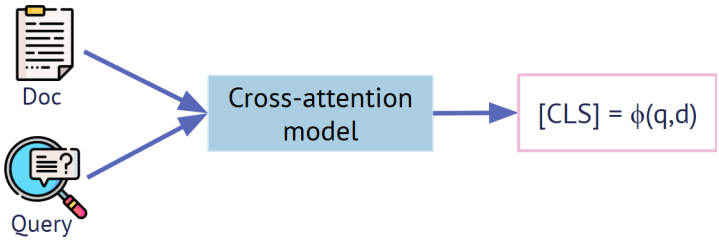


Figure 2.3: Cross-encoder computes similarity score $\phi_D(q, d)$ via cross-attention.

Traditional re-rankers employed cross-encoders with large contextual models [46, 45]. The computational costs for re-ranking inference are proportional to the sparse retrieval depth k_S, because the model has to process each query-document pair. Chunking documents into passages enlarges the number of pairs even further. Low inference latency is essential to many applications; we therefore focus on the more efficient dual-encoders instead (subsection 2.4.2).

BERT

In 2018, Google published the Bidirectional Encoder Representations from Transformers (BERT) model [15]. It uses a WordPiece tokenizer [87], and as a Transformer (section 2.3), it employs advanced techniques such as self-attention and multi-head attention [81, 2] to capture relationships between words by *bidirectionally* considering their preceding and succeeding words. This open-source large language model revolutionized neural model research in IR and NLP [46, 63, 76]. Most dense ranking models today are BERT-based [46, 45]. The full BERT architecture is visualized in Figure 2.4

BERT is trained on masked language modeling (MLM) and next sentence prediction (NSP) [15]. In MLM, some words are masked with $[MASK]$ tokens and the model learns to predict those words. In binary NSP, the model learns whether a sentence logically follows from another.

Reflecting on the name, *"Bidirectional Encoder Representations from Transformers (BERT)":*
We explained the model's bidirectional attention above. Furthermore, BERT adopts a Transformer architecture (Figure 2.2) without any decoder layers varying parameters for: L encoder layers, H hidden dimensions, and A attention heads. These variables are also included in Figure 2.4. The main BERT variants are $\text{BERT}_{\text{BASE}}$ ($L = 12, H = 768, A = 12$) and $\text{BERT}_{\text{LARGE}}$ ($L = 24, H = 1024, A = 16$).

Given a BERT-based encoder and a query q, the query representation is computed as:

$$\zeta^{BERT}(q) = BERT_{CLS}([CLS]\ t_1\ \cdots\ t_{|q|}\ [SEP]), \tag{2.7}$$

where $BERT_{CLS}(x)$ indicates a forward pass through the BERT model using input x and returns the vector corresponding to the $[CLS]$ classification token in the output layer; and the $[CLS]$ and $[SEP]$ tokens are appended to the input tokens.

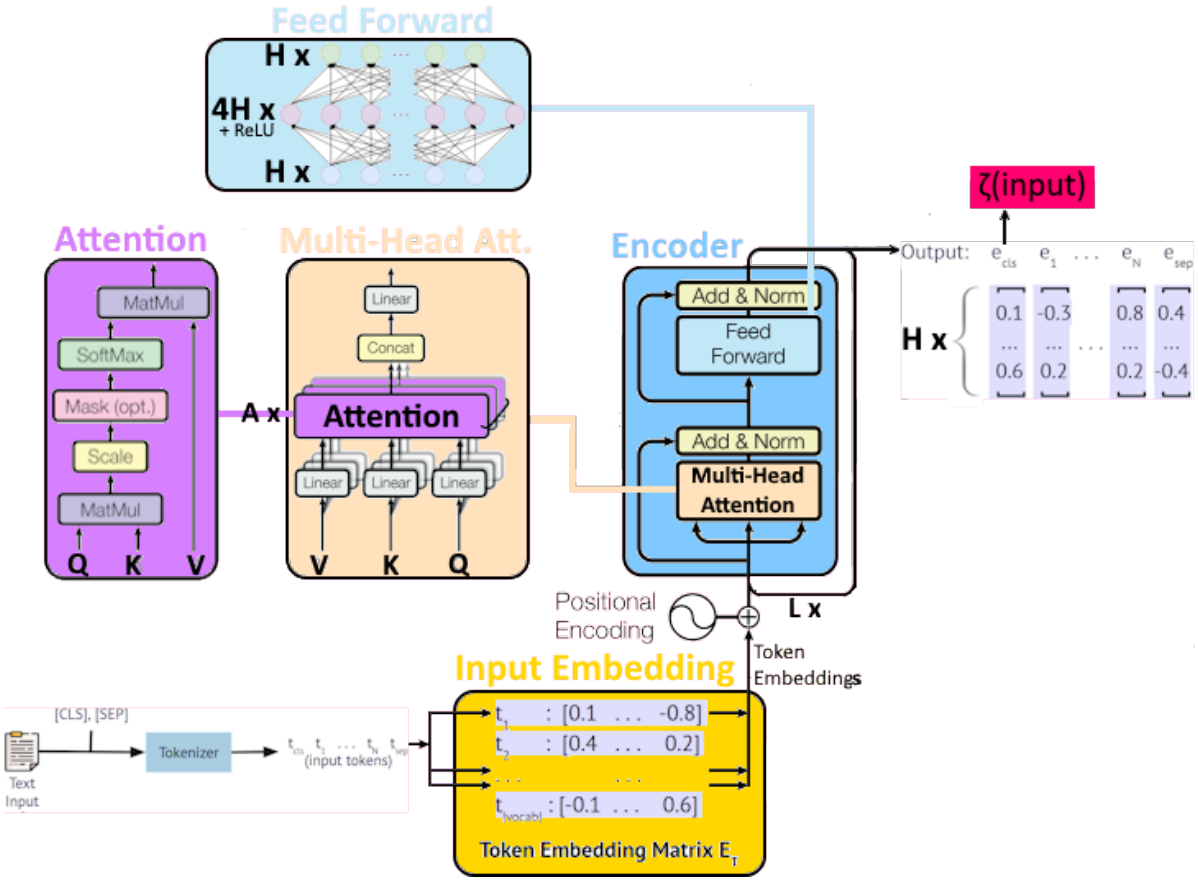


Figure 2.4: BERT [15] architecture. Partly copied from Vaswani et al. [81] but adjusted and extended to increase interpretability. Differences to regular Transformer architecture (Figure 2.2): [CLs] and [SEP] tokens are added to the input; no decoder layers; final representation comes from the output embedding vector $e_{[CLS]}$.

MonoBERT

A popular BERT adaptation for cross-encoding is MonoBERT [64], which fine-tunes BERT on a binary text classification problem. The model input is a query-document pair where q and d are truncated to 64 and 512 tokens respectively and concatenated as "$[CLS]\ q_1\ \cdots\ q_N\ [SEP]\ d_1\ \cdots\ d_M\ [SEP]$, where N and M denote the query and document lengths. The relevancy score is computed by parsing the BERT [CLS] token through a multi-layer perceptron (MLP) [67] into a float score between 0 (not relevant) and 1 (relevant). Finally, the pairs are ranked according to their scores.

2.4.2. Dual-Encoders

A second approach to neural ranking is dense passage retrieval (DPR) [36], which employs a dual-encoder architecture in which two independent semantic models respectively embed the queries and documents into a common vector space [60], and the similarity scores $\phi_D(q, d)$ are then computed based on similarity metrics between those vector representations. This approach is visualized in Figure 2.5. These encoder models often employ BERT-based models, resulting in 768-dimensional vector spaces [36, 37, 88, 46].

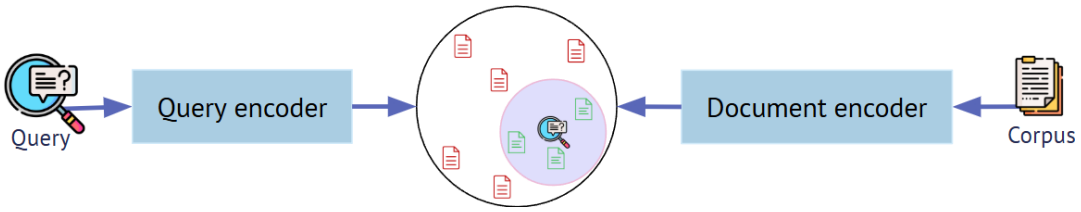


Figure 2.5: Dual-encoder computes similarity score $\phi_D(q, d)$ as distance between vector representations.

The dual-encoder structure is efficient because the document embeddings can be pre-computed and indexed in an offline phase called *indexing* [34]. This index is then loaded into RAM before runtime [46], trading memory space for efficiency and making it a viable option for both retrieval and re-ranking. Passage ranking requires larger indexes to store all passage embeddings.

Dense retrieval is performed as k-nearest neighbor search (kNN), which is made more efficient with approximate nearest neighbor search (ANN) [57], GPUs [34], or reducing the overall index size [59, 33, 25]. Our research is focused on re-ranking the provided query-passage pairs at $k_S = 1000$ instead.

A common similarity score function is the *dot product* [46]:

$$\phi_D^{Dual-encoder}(q, d) = \zeta(q) \cdot e_d, \tag{2.8}$$

where ζ is the query encoder and e_d is the indexed document vector.

Most dual-encoders are either homogeneous [37, 36, 69] or Siamese [70, 50], utilizing symmetric query and document models or even sharing model weights. Queries are usually short [35] and can be computed on CPUs [45], provided that sufficient memory is available. They may not require the same complex models. Other approaches are distilled models (subsection 2.5.1), semi-Siamese models [35], or diverging from dense query representations altogether [90, 89], though homogeneous BERT-based models remain the most common choice.

Our research employs a dual-encoder architecture for passage re-ranking. Dual-encoders are more efficient and allow a much higher retrieval depth k_S than cross-attention models; our scoring consists entirely of index look-ups, query encoding, and dot product computation. The arbitrary search queries are unknown in advance and must be encoded online, forming a latency bottleneck; our research is focused on lightweight query encoding.

ColBERT

A widely popular dual-encoder is the model Contextualized Late Interaction over BERT (ColBERT) [37], which adapts a homogeneous BERT architecture. The query encoder receives an input "$[Q]\ q_1\ \cdots\ q_N$" and the document encoder "$[D]\ d_1\ \cdots\ d_M$". In both encoders, these inputs are linearly processed by BERT, propagated through a convolutional neural network (CNN) [65], and normalized.

Furthermore, ColBERT introduces the *late interaction* architecture, illustrated in Figure 2.6, which combines the efficiency of representation learning dual-encoders with the accuracy from cross-attention in cross-encoders. In this paradigm, most of the query-passage interaction remains separated, though

the scoring function *MaxSim* still computes a token-level *fine-grained similarity* between the two:

$$\phi_D^{MaxSim}(E_q, E_d) = \sum_{e_q \in E_q} \max_{e_d \in E_d} \left(e_q \cdot e_d^T\right), \tag{2.9}$$

where E_q and E_d denote all embedded q and d tokens in the final BERT layer.

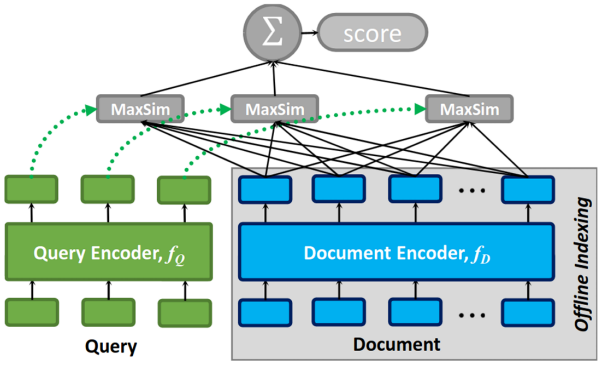


Figure 2.6: ColBERT's late interaction architecture, copied from the original paper [37].

All document token embeddings are pre-computed and indexed. Inference consists of: query encoding, computing the dot products between all query tokens and document tokens, and summing their similarities. All online computations except for the query encoding are lightweight and efficient. More recent research introduced TCT-ColBERT, a distilled ColBERT model that replaces its late interaction with representation learning. This model is described in subsection 2.5.1 and used as our baseline.

AvgTokEmb

A more recent study by Leonhardt et al. [45] introduces another lightweight approach to query encoding. In the re-ranking setting with score interpolation and a representation learning dual-encoder, they introduce an efficient *embedding-based query encoder* accompanied by a BERT-based document encoder.

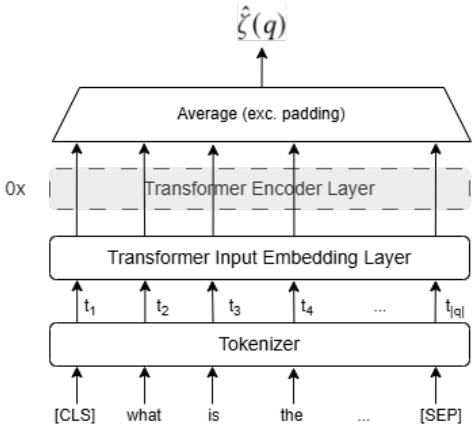


Figure 2.7: AvgTokEmb query encoder; defined by Leonhardt et al. [45] as "embedding-based query encoder".

The embedding-based query encoding architecture originates from a Transformer architecture (Figure 2.2) [81] but is much simpler, and visualized in Figure 2.7. The queries are first traditionally transformed by a WordPiece tokenizer [87] into BERT input sequences: "$[CLS]\ q_1 \cdots q_N\ [SEP]$". They are then only parsed through the BERT input embedding layer, skipping any self-attention transformer encoder layers, hence the name. The query representation is computed as the average of all query

token embeddings:

$$\zeta^{AvgTokEmb}(q) = \frac{1}{|q|} \sum_{t \in q} e_t, \tag{2.10}$$

where e_t is the embedding vector of a token t, retrieved by a look-up in the token embedding matrix $E_T \in \mathbb{R}^{vocab\_size \times emb\_dim}$; $vocab\_size$ is the tokenizer vocabulary size, i.e. 30522 for BERT's WordPiece tokenizer [87, 15]; and emb_dim is the document embedding dimension, i.e. 768 for BERT-based models [15].

The similarity score is a dot product between the query and document embeddings, as described in Equation 2.8. This encoding disregards any computationally expensive self-attention and relies solely on efficient token embedding look-ups e_t and simple calculations for averaging and dot products. This lightweight approach allows faster computations even on CPUs.

To avoid naming confusion with our proposed model, and because it relates specifically to averaging token embeddings, we refer to their embedding-based encoder as *average token embedding encoder (AvgTokEmb)*. Our proposed model is designed in the same re-ranking setting and integrates AvgTokEmb query encoding as an integral part of its architecture. Consequently, AvgTokEmb serves as one of our baselines for performance and efficiency.

2.5. Training AI Models

An early stage in an artificial model timeline is *training*, where the model learns to adjust its weights to perform better at a task [26]. Afterwards, these trained models can be applied to their respective tasks, such as indexing and passage re-ranking.

AI models are initialized with random or pre-trained parameters [15, 46]. Training is then performed in a training loop [17], consisting of:

1. A forward pass where the input data is passed through all model layers to generate predictions.
2. Loss calculation in which the model performance is evaluated against actual target values using a loss function [82].
3. Backpropagation [85, 4] where the computed loss is propagated backwards through the model and loss gradients with respect to the model's parameters are calculated using the chain rule.
4. Updating the model parameters using an optimization algorithm such as stochastic gradient descent (SGD) [4] or Adam [39]. This makes the model predictions more accurate.

The model is periodically evaluated during training in a process called *validation*, checking the predictions based on a label representing the truth. This helps identify underfitting [32], where a model fails to capture the underlying data patterns because it is either too simple or incorrectly designed. The stopping condition is either a defined number of passes through the dataset (epochs) or an early stopping criterion to prevent overfitting [32] on the training data and losing its general reasoning capabilities.

Effective machine learning models are typically trained in two phases: pre-training and fine-tuning. In pre-training, a model learns a general language understanding from a vast and diverse dataset. It is infeasible for most individuals due to the immense data and computational power requirements. The most popular pre-trained model is Google's BERT [15], which instigated many other neural IR research [64, 36, 37, 46]. Fine-tuning, on the other hand, takes a pre-trained model and adapts it for a specific task or domain using a smaller, more focused dataset. This phase updates the model's weights to enhance its performance in specific contexts, such as passage ranking, where models are often fine-tuned on large labeled datasets like MS MARCO [62]. Together, pre-training and fine-tuning allow models to leverage general knowledge and apply it effectively to specialized tasks, creating versatile and powerful AI systems.

2.5.1. Knowledge Distillation

An example of a related process to fine-tuning is knowledge distillation (KD) [28, 46], a technique where a student model is trained to replicate the behavior of a powerful teacher model. This process, illustrated in Figure 2.8, encourages the student model to make high-quality predictions that mimic the teacher model outputs while being more efficient overall.

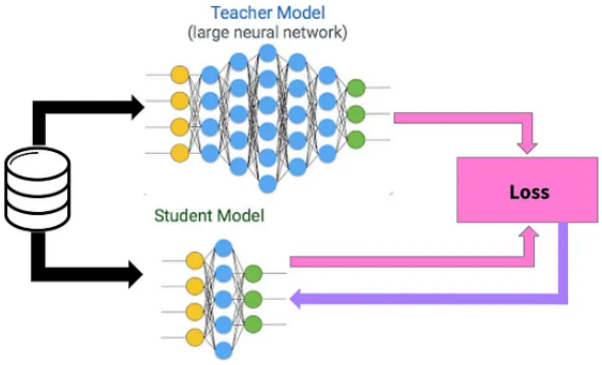


Figure 2.8: Knowledge distillation architecture, copied from Ganesh [22].

The primary goals of KD are reducing inference latency or index storage space or increasing performance. For instance, techniques like recurrent neural networks (RNNs) are suitable for handling short queries incrementally as they are being typed [9]. Notable examples of distilled IR models include the general-purpose DistilBERT [74] and EmbedDistill [38].

Techniques like the Margin-MSE loss [31] optimize the margin between relevant and non-relevant passages, addressing cross-architectural issues and allowing an ensemble of teachers [30]. Furthermore, RocketQA [69] introduces a unified distillation method to simultaneously train for DPR and passage re-ranking.

Hard negative samples play an important role in effective model training. These are challenging non-relevant examples that are difficult to distinguish from relevant examples. Such techniques include in-batch negatives [49] and Balanced Topic Aware Sampling (TAS-B) [30, 84], in which passages are clustered on relevancy and sampled balanced on their pairwise margins.

SPLADE++ (CoCondenser-SelfDistil)

The original SPLADE model [18], described in section 2.2, was trained end-to-end in a single retrieval stage with in-batch negatives. In 2022, SPLADE++ [19] extended this research with alternative methods that include knowledge distillation, hard negative sampling, and initialization from a pre-trained CoCondenser [23], making the sparse neural model training more effective.

The SPLADE research has since been extended with an efficiency study [41] that achieves similar latency to BM25 with <10% performance loss. Additionally, SPLADE-v3 [42] extended training with KL-Div [77] and MarginMSE [31].

We compare our model against the SPLADE++ CoCondenser-SelfDistil model as a learned sparse retrieval baseline, as it achieves the highest performance among sparse retrieval methods in Pyserini [47].[2] We abbreviate this model to SPLADE++ CC-SD.

[2]Pyserini reproductions: https://castorini.github.io/pyserini/2cr/msmarco-v1-passage.html

TCT-ColBERT

ColBERT, introduced in subsection 2.4.2, has the main disadvantage of high memory requirements because each passage token embedding is stored individually. TCT-ColBERT [48] attempts to mitigate this by returning to representation learning and dot product similarity as described in Equation 2.8, while distilling the late-interaction performance using an approach called *tightly coupled teachers (TCT)*. In 2021, Lin, Yang, and Lin [49] improved TCT-ColBERT training with in-batch negatives.

In traditional distillation methods, the scores for queries and documents are pre-computed by the teacher model [31]. In the proposed TCT approach, the teacher model itself is incorporated in the student model training to provide real-time guidance. It defines two probability functions, respectively computed by the ColBERT teacher and TCT-ColBERT student, and trains the student on KL-Div [77] between the probabilities. This method enables more flexible distillation strategies, leading to better learned representations and reduced query encoding latency. Additionally, it greatly reduces storage requirements and removes the need for periodic index refreshes during representation learning [88]. A practical consequence of this setup is that the teacher model itself must be reasonably efficient, ruling out cross-encoders.

Our model is distilled on query representations encoded by TCT-ColBERT. Additionally, we use TCT-ColBERT as a baseline in three settings: dense retrieval, re-ranking, and interpolated re-ranking.

3

Average Embedding Query Estimation

As introduced in chapter 2, this research is focused on the efficiency and performance of the neural re-ranking. The starting point for this thesis is the recently published paper by Leonhardt et al. [45]. Their research employs a dual-encoder as a re-ranker, which uses two individual query document encoders, allowing document embeddings to be pre-computed and stored as indexes. These indexes are loaded into random access memory (RAM) before inference. The re-ranking scores are computed as a simple score interpolation between the lexical sparse scores $\phi_S(q, d)$ from first-stage retrieval and the semantic dense scores $\phi_D(q, d)$ from re-ranking (Equation 2.2). The score interpolation is validated with a hyperparameter α, ranging from 0 to 1, which determines the impact of each score. In turn, $\phi_D(q, d)$ is computed as the dot product between the query and document vectors (Equation 2.8). This initial architecture is visualized in Figure 3.1.

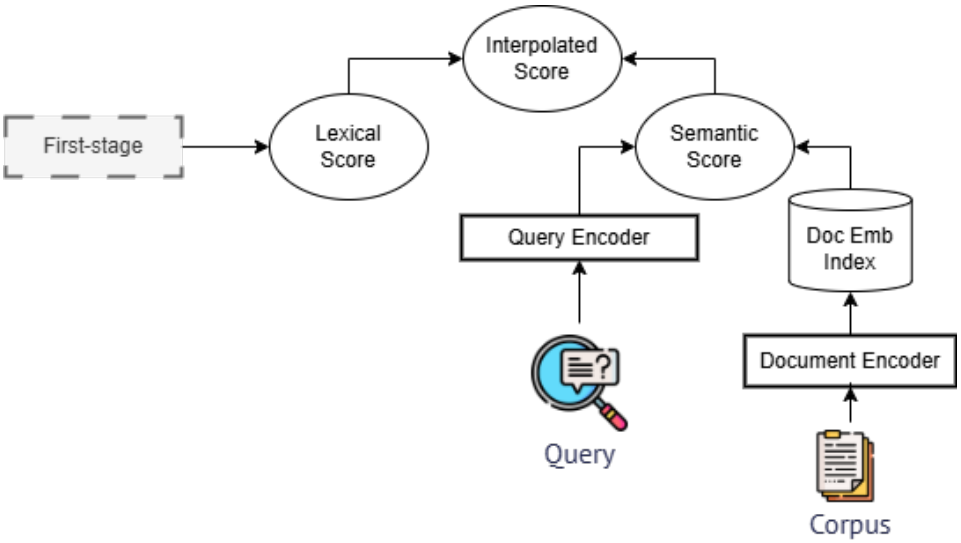


Figure 3.1: Original architecture of neural re-ranking using Fast-Forward Indexes

Our foundational insight is that the query vector resembles its most relevant document embeddings. Those query-document pairs achieve the highest similarity scores from dot products, meaning that their representation vectors are similar, which allows us to approximate this vector using low-latency operations while removing the need for neural query encoding via attention mechanisms [15, 81].

To reiterate, our research focus applies to efficiently creating a query representation in dual-encoder re-ranking. This section provides a detailed examination of our methodology and approaches. These methods are clarified and motivated through various visualizations and algorithms, enhancing the comprehensibility and interpretability of each concept. section 3.1 introduces our proposed query encoder model and explains the reasoning and intuition behind the model design choices. The training architecture is discussed in section 3.2. section 3.3 concludes the chapter by addressing some shortcomings and suggesting alternative approaches.

3.1. Model

The intuition behind the model design is best explained by an iterative reasoning example, guided by a visualization. This oversimplified example is introduced in Figure 3.2a, please read the caption carefully before continuing. Note that this is a simplified 2-dimensional example with only 13 documents, though this intuition behind dot product similarity scales perfectly to vectors of large dimensions [3] and many documents, such as an index of 768-dimensional BERT-based passage embeddings [46, 45].

3.1.1. Averaging over Top-Ranked Document Embeddings

In order to retrieve all green documents in Figure 3.2a, the query vector $\zeta(q)$ would be estimated somewhere between all green documents. Initially during re-ranking, the encoder is only provided with a lexical ranking. Therefore, research question 1 excludes any semantics and focuses solely on the lexically most relevant documents. We propose a novel initial model named the Average Embedding ($\text{AvgEmb}_{\text{n-docs}}$) query estimator, which assigns a weight to the vectors of the n lexical top-ranked documents according to their rank, and then predicts the query vector $\zeta(q)$ as the weighted average representation of these vectors:

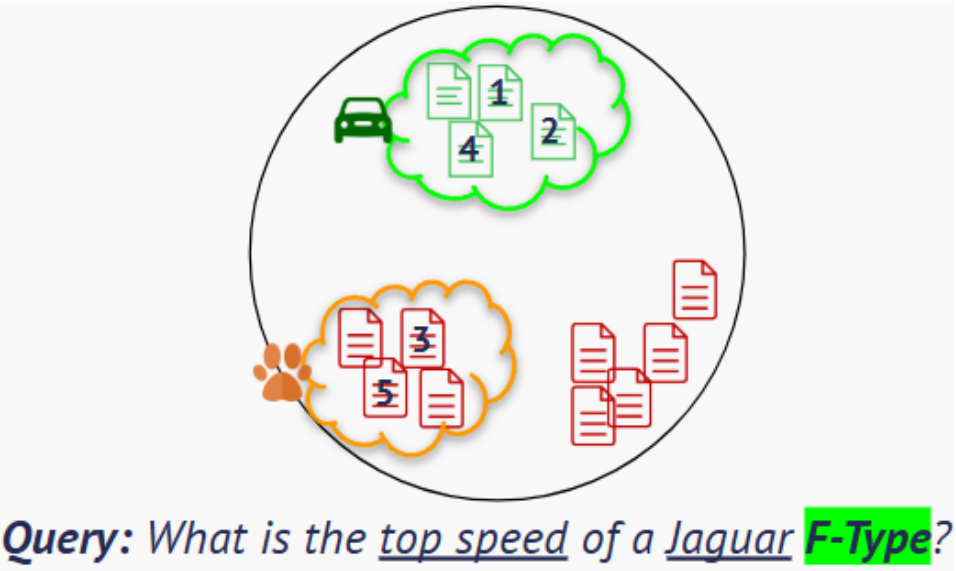
$$\zeta^{AvgEmb_{docs}}(q) = \sum_{d_i \in R(q)} \omega_i \cdot e_{d_i}, \tag{3.1}$$

where $R(q)$ denotes the lexical ranking for query q; d_i represents the document at rank i; e_{d_i} is the embedding of d_i as retrieved from the index; and w_i denotes the weight assigned to the i-th embedding, normalized such that $\sum_{d_i \in R(q)} w_i = 1$.

In this specific re-ranking setting with dual-encoders and pre-computed in-memory document embeddings, both the lexical ranking and all document vectors are available without additional computational costs. The proposed estimation method only involves embedding look-ups and weighted averaging. It does not require any complex self-attention encoder layers or expensive calculations. In theory, this leads to an extremely fast query encoding during inference.

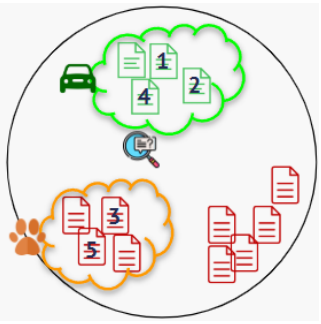
This model definition raises 2 important sub-questions to optimize model efficiency and performance:

- *"How should the embedding weights be distributed?"*
  This will mostly be answered in our continuous example. Alternative weighting methods such as probability distributions are explored in subsection 5.2.2.
- *"What is the optimal amount of documents to average over?"*
  This is most effectively answered by experiments, explored in subsection 5.2.1.



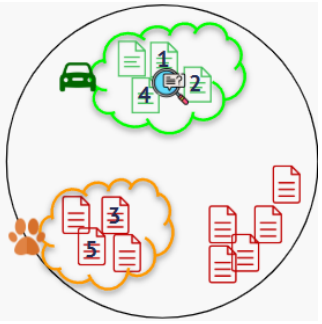
(a) No estimation

	w_i
d_1	0.2
d_2	0.2
d_3	0.2
d_4	0.2
d_5	0.2



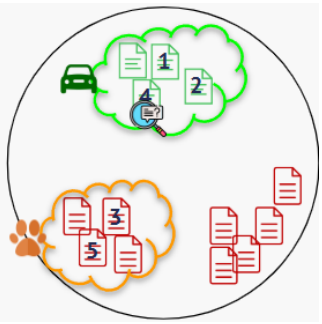
(b) Uniform weights

	w_i
d_1	0.35
d_2	0.3
d_3	0
d_4	0.25
d_5	0



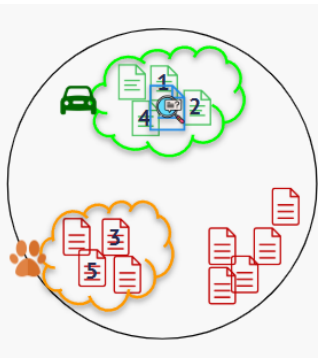
(c) Optimal weights for specific query

	w_i
d_1	0.3
d_2	0.25
d_3	0.2
d_4	0.15
d_5	0.1



(d) Learned weights

	w_i
q_{light}	0.5
d_1	0.15
d_2	0.125
d_3	0.1
d_4	0.075
d_5	0.05



(e) Learned weights with lightweight query-encoding

Figure 3.2: Simplified 2D vector dual-encoder index, introducing the `AvgEmb` estimator approach. The query q is *"What is the top speed of Jaguar F-Type"*, an instance of the vocabulary mismatch problem [21]. Relevant documents are **green**, non-relevant documents are **red**. The green cloud contains relevant documents (about Jaguar cars). The orange cloud contains documents (about Jaguar animals) that have high lexical scores $\phi_S(q, d)$ but low semantic scores $\phi_D(q, d)$. The numbered documents represent the lexical ranking for q, with numbers i representing the ranking of document d_i. Figures b-e include a weight table denoting weight w_i for the listed embeddings.

Uniform Averaging

A simple and intuitive method for distributing weights across documents involves assigning uniform weights and computing the average vector representation $\zeta(q)$, equivalent to Equation 2.10:

$$\zeta^{uniform}(q) = \frac{1}{|q|} \sum_{t \in q} e_t, \tag{3.2}$$

where e_t denotes the embedding of token t in q.

Consequently, the query vector is approximated as the exact centroid of its n lexically most relevant documents, as depicted in Figure 3.2b. The corresponding weights for each vector are illustrated in the table on the left. The magnifying glass icon indicates the query vector estimation.

A quick assessment of the example already reveals that this method is overly simplistic and does not yield an accurate query estimation, as the non-relevant documents from the lexical ranking impact the outcome prediction too heavily.

Weighted Averaging

In reality, the query exhibits greater resemblance to semantically relevant documents than others. For instance, the example query only concerns cars, and documents on the Jaguar animal are irrelevant. To address this, we propose employing weighted averages rather than uniform weights:

$$\zeta^{weighted}(q) = \sum_{t \in q} \omega_t \cdot e_t, \tag{3.3}$$

where e_t again represents the embedding of token t; and ω_t denotes the weight assigned to query token t, normalized such that $\sum_{t \in q} \omega_t = 1$.

As demonstrated in Figure 3.2c, weights can be distributed optimally for the query by assigning weights of 0 to semantically non-relevant documents d_3 and d_5. The remaining weights are allocated to genuinely relevant documents.

Learned Weights

The primary limitation of the preceding example is that such exact optimization is only possible for a specific query alone. It does not generalize well to a search engine designed to handle arbitrary search queries.

This necessitates a model capable of learning a weight distribution that performs consistently well across diverse inputs. These learned weights are expected to diminish as the query-document relevancy decreases. Therefore, we hypothesize that the learned weights naturally decrease along the document ranks, given that the first document is generally more relevant than the second, and so on. A detailed training description for the embedding weights is provided in section 3.2.

Figure 3.2d illustrates that such learned weights on the lexical ranking alone do not produce an ideal estimation. However, the approach only leverages lexical information so far, leaving it vulnerable to the vocabulary mismatch problem [21]. We therefore speculate that incorporating semantical information about the query is essential for accurate predictions. subsection 3.1.2 explores our approach for integrating semantics into the model.

Insufficient Embeddings Problem

Although more importantly, this framework raises an issue that should be addressed: The provided BM25 ranking does not consistently retrieve at least n_docs documents for each query. This issue arises due to BM25's exact-term matching nature and is further accelerated by stopword removal prior to this matching. For instance, the query "what is theraderm used for" is reduced to just "theraderm", which is an uncommon term and only occurs in five documents, causing problems for an $\text{AvgEmb}_{\text{10-docs}}$ model. Our initial model addresses this problem by only considering the retrieved documents and normalizing their accompanied weights.

However, a more severe issue arises when certain queries retrieve no documents at all. This would result in division by zero, which is mathematically impossible [61]. Luckily, this issue is indirectly addressed by the addition in the next subsection, which guarantees that at least 1 embedding exists.

3.1.2. Adding Query Semantics

The previous subsection introduced the importance of query semantics for accurate query representations. The query vector simply correlates more to semantically relevant documents, though lexical retrieval falls short in providing such information.

We propose a final estimator model, termed $\text{AvgEmb}_{\text{q,n-docs}}$, or simply AvgEmb. This model concatenates the document embeddings with a lightweight query vector q_{light}, which is assigned its own weight. Given that the lightweight query encoding integrates semantic information, we hypothesize that the training process will inherently assign it a large, significant weight. This new prediction is highlighted in Figure 3.2e, and demonstrates promising accuracy.

Achieving accurate encoding involves a balance between enhanced accuracy at the cost of increased latency. Integrating a regular query encoder into our model would substantially increase latency and render the document embeddings redundant. Consequently, we adopt a lightweight query-encoding approach instead.

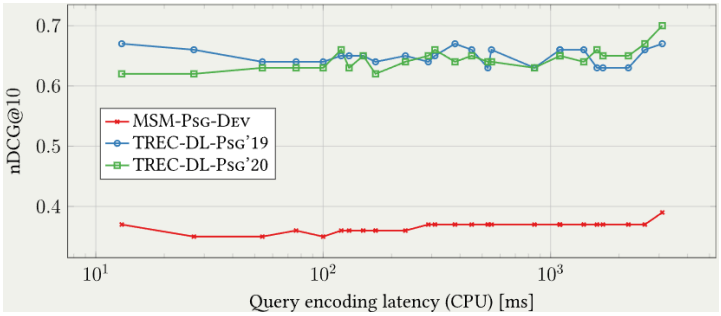


Figure 3.3: Performance-Efficiency comparison of different lightweight encoders on 3 test datasets. The left-most measurements are from an embedding-based transformer architecture. The others represent transformer-based architectures with varying amounts of hidden dimensions, encoder layers, and attention heads.
This figure is copied from Leonhardt [44], corresponding to the Fast-Forward indexes paper [45].

Prior research [45] has already explored some lightweight query-encoding options. The key findings are summarized in Figure 3.3. We are particularly interested in the left-most measurements in this graph, as they indicate the most significant efficiency gains with minimal performance loss. This model reflects the AvgTokEmb encoder outlined in subsection 2.4.2. The AvgTokEmb query-encoding framework is depicted in Figure 2.7.

Each query in the encoding batch is first transformed into query tokens. Shorter queries are padded with [PAD] tokens to match the length max_len of the longest query in the batch. The embedding layer is a quick embedding lookup in a matrix $\in \mathbb{R}^{vocab\_size \times emb\_dim}$, yielding one embedding of emb_dim elements for each individual token in the vocabulary. The padding token embeddings are masked, and the remaining token embeddings are uniformly averaged over. This model is fast due to its absence of

transformer encoder layers (and hence self-attention). It only consists of embedding look-ups and an averaging operation, similar to our AvgEmb model. Thus, our model maintains its high efficiency, even with the inclusion of the AvgTokEmb query-encoding.

Integrating the lightweight encoder into our AvgEmb model results in the query encoding algorithm depicted in Figure 3.4.

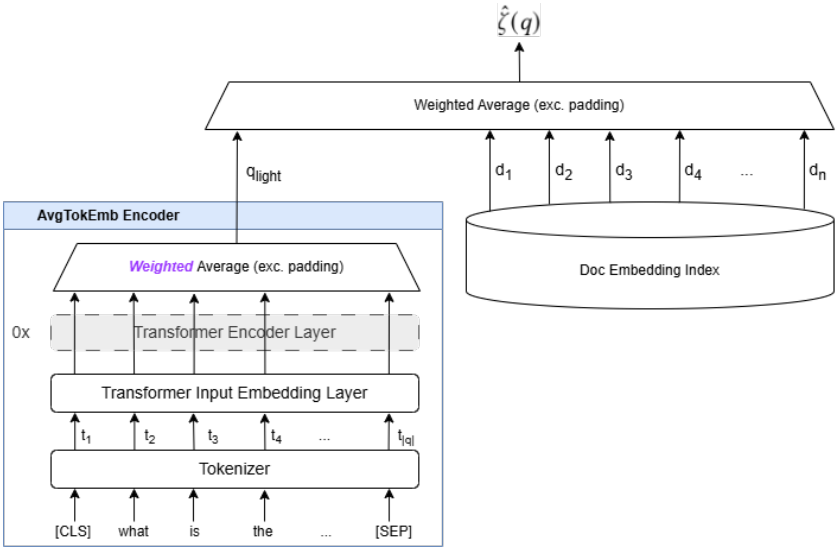


Figure 3.4: AvgEmb query encoding architecture after adding (weighted) lightweight query-encoding from AvgTokEmb.

Our updated estimator thus computes a query representation $\zeta(q)$ with this formula:

$$\zeta^{AvgEmb}(q) = \omega_q \cdot \sum_{t \in q} \omega_t \cdot e_t + \sum_{d_i \in R(q)} \omega_i \cdot e_{d_i}, \tag{3.4}$$

where operations related to q_{light} are highlighted in violet; $R(q)$ denotes the lexical ranking for query q; d_i represents the document at rank i; e_x is the embedding of $x \in \{\text{document } d_i, \text{query token } t\}$; and w_z denotes the embedding weight assigned to $z \in \{\text{query } q, \text{document } d_i \text{ at rank } i, \text{query token } t\}$, normalized such that $\omega_q + \sum_{d_i \in R(q)} w_i = 1$ and $\sum_{t \in q} w_t = 1$.

Remember that we incorporate this encoder into our dual-encoder with score interpolation. The resulting final similarity score $\phi(q, d)$ is thus computed as:

$$\begin{aligned}
\phi(q,d) &= \alpha \cdot \phi_S(q,d) + (1-\alpha) \cdot \phi_D(q,d), && \leftarrow \textit{Equation 2.2} \\
\phi_S^{BM25}(q,d) &= \sum_{t \in q} \frac{tf_{t,d} \cdot (k_1+1)}{tf_{t,d} + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avgdl}\right)} \cdot log\left(\frac{|C| - df_t + 0.5}{df_t + 0.5}\right), && \leftarrow \textit{Equation 2.4} \\
\phi_D^{Dual-encoder}(q,d) &= \zeta(q) \cdot e_d, && \leftarrow \textit{Equation 2.8} \\
\zeta^{AvgEmb}(q) &= \omega_q \cdot \sum_{t \in q} \omega_t \cdot e_t + \sum_{d_i \in R(q)} \omega_i \cdot e_{d_i}, && \leftarrow \textit{Equation 3.4}
\end{aligned} \tag{3.5}$$

where all notation rules from the original equations still apply.

Note that the (weighted) averaging approaches both require masking and normalization of the weights. The complete pseudocode for the query encoder is presented in Algorithm 1. In an effort to make the mathematical concepts more comprehensible, a simplified example iteration of the forward method is provided in Figure B.1.

Algorithm 1 AvgEmb Estimator class [code]

Function compute_query($embs$, $weights$, $mask$):
    $weights \leftarrow softmax(weights)$
    $weights \leftarrow weights * mask$
    $weights \leftarrow normalize(weights)$
    return $weighted\_sum(embs * weights)$

Function forward($queries$):
    $q\_tokens \leftarrow tokenize(queries)$ ▷ Gives $input\_ids$, $attention\_mask$, max_len
    $q\_tok\_embs \leftarrow get\_tok\_embs(input\_ids)$ ▷ Shape ($batch$, max_len, dim)
    $q\_tok\_weights \leftarrow tok\_embs\_weights[input\_ids]$ ▷ Shape ($batch$, max_len)
    $q_{light} \leftarrow$ compute_query($q\_tok\_embs, q\_tok\_weights, attention\_mask$)
    $top\_docs\_ids \leftarrow sparse\_index.transform(queries)$
    $top\_docs\_embs \leftarrow index.get\_vectors(top\_docs\_ids)$ ▷ Shape ($batch$, n_docs, dim)
    $embs \leftarrow q_{light} :: top\_docs\_embs$
    $embs\_weights \leftarrow embs\_weights\ \forall queries$ ▷ Shape ($batch$, n_embs)
    $embs\_mask \leftarrow mask\_top\_docs()$ ▷ Shape ($batch$, n_embs)
    $q\_estimation \leftarrow$ compute_query($embs, embs\_weights, embs\_mask$)
    return $q\_estimation$

3.1.3. Refining the Architecture

An initial architecture combining the AvgEmb and AvgTokEmb encoders is illustrated in Figure 3.5. The key limitation of this design is that the AvgTokEmb encoder is fine-tuned separately from the AvgEmb estimator, in an independent dual-encoder training setup. This necessitates its own document encoder and index combination. This results in double the index storage space, index operations, re-indexing, and score interpolation.

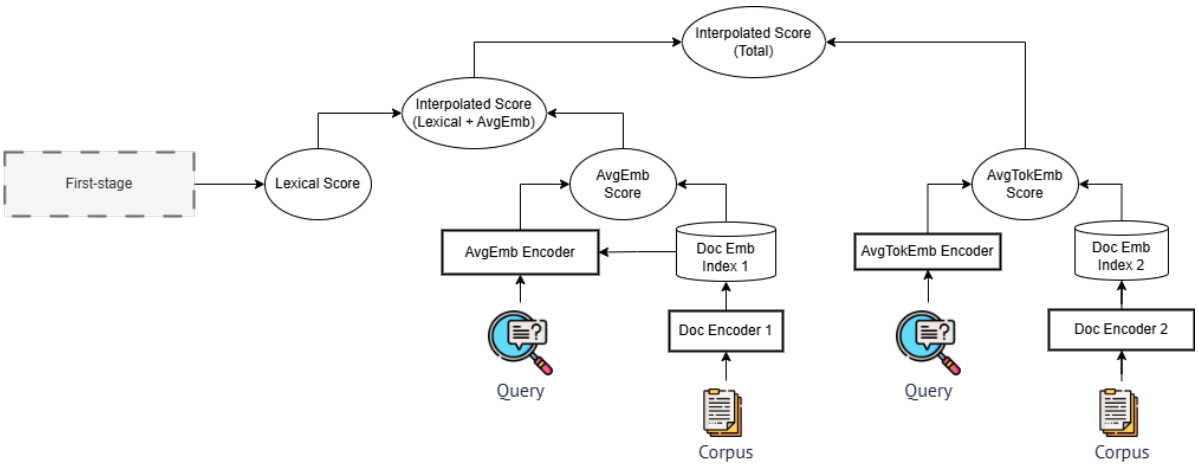


Figure 3.5: Initial architecture after combining AvgTokEmb and AvgEmb encoders.

In an improved architecture, the AvgEmb estimator integrates the AvgTokEmb encoder, and the combination is trained to predict the same document vectors. This reduces the architecture complexity significantly to the design illustrated in Figure 3.6.

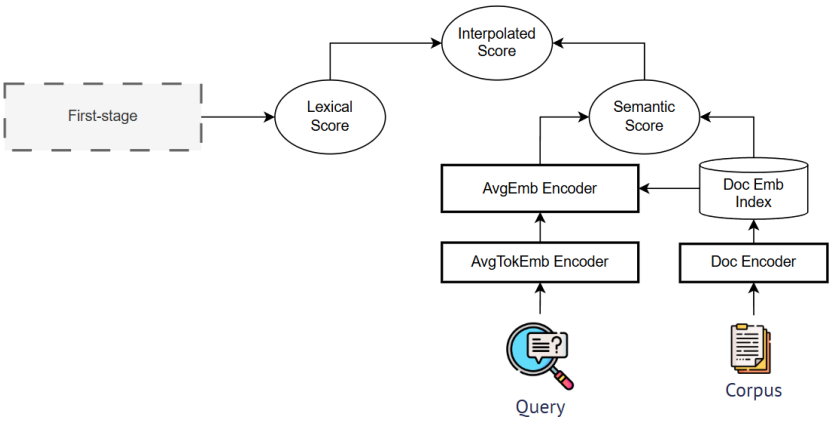


Figure 3.6: Refined architecture after combining AvgTokEmb and AvgEmb encoders.

3.2. Training

The previous section described the necessity of learning optimized model weights through rigorous AI model training. The fundamental principles of AI model training are detailed in section 2.5.

The model architecture incorporates three variables that require training or fine-tuning:

- $E_T \in \mathbb{R}^{vocab\_size \times emb\_dim}$:
  A matrix that provides embeddings for each query token, indexed by the token ID. $vocab\_size$ denotes the tokenizer vocabulary size, e.g. 30522 for BERT's WordPiece [87, 15]; and emb_dim represent the document encoder's embedding dimension, e.g. 768 for BERT [15]. For instance, $e_{101} \in E_T$ yields an embedding for the [CLS] token with ID 101.
- $W_{tok\_embs} = \{\omega_t \mid t \in \text{vocabulary}\}$:
  A 1-dimensional vector of uniformly initialized weights, to assign weights ω_t to each token embedding in the q_{light} computation, intended to distinguish reward context-defining tokens (e.g. "why", "capital") and reduce the impact of less important tokens (e.g. stopwords such as "the").
- $W_{embs} = \{\omega_i \mid i \in \{0 \text{ (query)}, \text{ lexical rank } i\}\}$:
  Another 1-dimensional vector of uniformly initialized weights, to assign weights (ω_q, ω_{d_i} to each embedding during the final weighted averaging computation.

The forward method of any (BERT-based) query encoder model processes batches of queries and returns a batch of query encodings. The employed loss function is mean squared error (MSE) loss [79]:

$$\mathcal{L}^{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3.6}$$

where y_i represents the teacher encoding; $\hat{y}_i$ is the teacher encoding; and n is the number of $(y_i, \hat{y}_i)$ pairs over which the summation is computed.

This training regime aims to accurately predict similar representations as the teacher encodings, effectively establishing a knowledge distillation framework where the AvgEmb estimator acts as the student model, and TCT-ColBERT serves as the teacher model.

The training architecture is depicted in Figure 3.7. The student model is essentially guided to create query representations that resemble the teacher's encodings. Since each training iteration requires the

same data, it is beneficial to pre-compute the teacher encodings and store them in a reusable Query Embedding Index. Additionally, the lexical ranking over all training and validation data can also be pre-computed and stored.

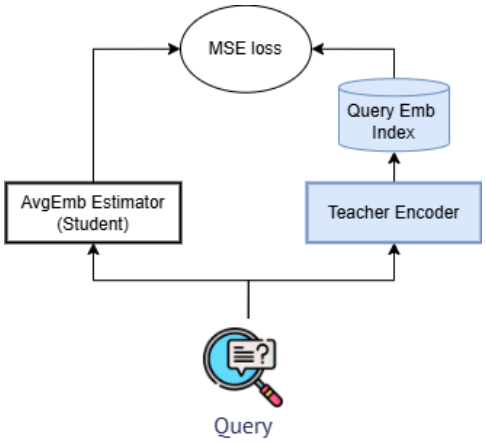


Figure 3.7: AvgEmb estimator training architecture (MSE loss)

Unfortunately, further refinements were not possible due to time constraints. Therefore subsection 3.3.1 outlines the existing limitations and proposes more complex improvements for future work.

3.3. Alternative Choices

The third research question delves into the limitations of this research and explores alternative methods and settings that could potentially boost the model performance and efficiency. Specifically, it aims to identify approaches that address shortcomings in the current methodology and provide a comprehensive analysis of said potential improvements. By examining these alternatives, the research endeavors to contribute valuable insights that could pave the way for more robust and efficient information retrieval systems.

subsection 3.3.1 discusses the most noteworthy shortcoming, namely the trivial and restrictive model training. subsection 3.3.2 examines the `AvgTokEmb` lightweight query encoder in more detail.

3.3.1. Training Alternatives

Different training approaches may yield better results. This subsection describes some downsides to the current training setup and suggests some alternatives to reach the full potential of the model. Unfortunately, these proposed improvements for training could not be carried out within the thesis timeframe, partly due to my novelty with these subjects. Consequently, I outline the theoretical framework for future research endeavors to address these limitations.

One minor downside to our model design is that our approach only assigns weights to document ranks d_i for the final weighted averaging, and does not directly exploit differences in lexical scores between documents d_i and d_{i+1} for a specific query. This is only relevant when the lexical ranking exhibits an unusually large relevancy gap between scores in subsequent documents. For instance, the relevancy scores 30, 29, 15, 14, 13 contain a relatively big gap between d_2 and d_3 which would be ignored in the current approach. Though we are unsure how often such inconsistencies appear in practice.

However, the primary issue is that the existing training configuration (described in Figure 3.7) essentially teaches the `AvgTok` estimator to predict a vector that resembles the teacher encoding. The model is trained using Mean Squared Error (MSE) loss between the student and teacher query encodings. Consequently, the teacher model's performance sets a ceiling on the optimal performance of the student model. The following subsections detail iterative improvements to the training setup.

Dual-encoder training

The first suggestion is to train a query encoder in parallel with a corresponding document encoding, alas a dual-encoder training approach. This configuration allows validation on (re-)ranking loss or other retrieval metrics. The model weights can be initialized from pretrained models such as `bert_base_uncased` [15], and both encoders can be fine-tuned in tandem. Because the document encoder now also undergoes fine-tuning, it is no longer feasible to index the document encodings. Instead, the top-ranked document texts must be encoded on the fly.

This training architecture is similar to Figure 3.1. The model is fine-tuned on the RetrievalMAP metric during validation, with a maximum of 50 epochs and early stopping after 5 validations without any improvement in this metric.

This more sophisticated setup allows more advanced negative sampling and loss functions, resulting in more computationally efficient training. Dual-encoders for information retrieval typically leverage a contrastive loss function [36]:

$$\mathcal{L}(q, d^+, D^-) = -log\left(\frac{\exp(\phi(q, d^+; \theta)/\tau)}{\sum_{d \in D^- \cup \{d^+\}} \exp(\phi(q, d; \theta)/\tau}\right), \tag{3.7}$$

where a training instance consists of a query q, a relevant document d^+, and a set of irrelevant documents D^-; and τ is a hyperparameter controlling temperature.

The MSMARCO training set [62] contains only one document labeled as relevant per query. Therefore, negative samples can be created using in-batch negatives, which form D^- by incorporating all documents from other queries in the sampled set. The downside of this approach is that MSMARCO might have many false negatives that are relevant but labeled as such [69]. In-batch negative training with a false negative $fn \in D^-$ could adversely affect model performance.

Balanced Topic Aware Sampling (TAS-B)

Sampling hard negatives is important to enhance training efficiency for neural models. Hard negatives are non-relevant documents that share similar traits with relevant ones. By learning from hard negatives instead of randomly sampled negatives, the model is encouraged to identify more complex distinguishing features, thereby improving its general knowledge and performance. This method is particularly beneficial for effective training resource utilization, as it ensures consistent exposure to challenging scenarios, refining the model to make accurate predictions and reducing the need for large training batches.

Topic aware sampling (TAS) [30] generates hard negatives by clustering semantically similar queries together and sampling a batch from within one cluster. These query clusters are created once before training. The same paper introduces TAS-Balanced (TAS-B), which extends this idea with balanced margin sampling. In this approach, pairs of relevant/non-relevant passages are balanced by binned margins per query, mainly to down-sample large margin pairs without discarding any training data.

Teacher Ensemble

The third training suggestion is to optimize the margin between relevant and non-relevant sampled passages [31]. This alternative does not require the dual-encoder training setup and could be applied directly to the current training. For this to become feasible, a set of (preferably hard) negative documents should be created for each query in a batch. Scoring ranges do not matter when optimizing on MSE loss between encodings, only the relative differences. The loss function would be Margin-MSE loss [31]:

$$L(Q, P^+, P^-) = MSE(M_s(Q, P^+) - M_s(Q, P^-),\ M_t(Q, P^+) - M_t(Q, P^-)), \tag{3.8}$$

where batches contain triples of (queries Q, relevant passage P^+, and non-relevant passages P^-); M_s represents the student model and M_t the teacher model.

This loss function does not impose any assumptions on the model architectures, enabling the swapping in of various neural ranking models as teachers. These teacher scores can be pre-computed once and reused throughout training runs.

This framework enables training the student models on multiple teacher signals. The paper demonstrates that this further boosts model performance, without compromising efficiency since the underlying model architectures are not altered.

3.3.2. AvgTokEmb Exploration

Focusing on the model architecture again, the following improvements are proposed specifically for the `AvgTokEmb` encoder.

Stop Word Removal

Excluding semantically less significant query tokens before averaging these embeddings might be beneficial. Due to the arbitrary nature of queries, it is challenging to identify irrelevant tokens on a per-query basis. However, we can infer that common stop words like ”the” frequently occur in queries and should likely not have a significant influence on the retrieved documents.

Because these words are prevalent, they are likely trained as a general average embedding of all contexts they appear in. Stop word removal can be implemented by replacing stop words with padding tokens $[PAD]$ and adjusting the corresponding entries in the attention mask to zeros, excluding them from weighted averaging over the token embeddings. The PyTerrier Stopwords class[1] can serve as an example for such an implementation.

Additionally, it can be argued that stop word removal is partially achieved by learning distinct weights for each token embedding, as those tokens should ideally receive a low weight during fine-tuning.

Non-Fine-Tuned Token Embeddings

Understanding the concept of back-propagation in machine learning is essential for this alternative approach. Each training instance exclusively activates the tokens from queries within the batch and padding tokens. Consequently, tokens not present in the training data will not be fine-tuned, retraining their pre-trained model embeddings. Comparing a fine-tuned `AvgTokEmb` encoder with its pre-trained model initialization shows significant performance differences. This implies that non-fine-tuned token embeddings might negatively impact vector representations.

Our proposal was to track which tokens exist in the MSMARCO training data, and assign a lower weight to untrained tokens before weighted averaging. A preliminary experiment revealed that 26.249 tokens were encountered in the training data out of 30522 tokens in the BERT vocabulary size, amounting to a total of 86%. However, only the TREC DL 2020 [11] contained a single untrained token. Therefore, this experiment was discontinued as its impact on the test sets would not justify further pursuit.

Equivalent task as AvgEmb

An interesting observation is that the AvgTokEmb part of the model alone tries to predict the teacher representations as well. Therefore, when training AvgEmb, the weights W_{tok_embs} could be trained independently by returning q_{light} directly after computing it.

[1] PyTerrier Stopwords Class: http://terrier.org/docs/current/javadoc/org/terrier/terms/Stopwords.html

4

Experimental Setup

This chapter provides an overview of all details, requirements, and choices for the experiments in this thesis, which are presented in chapter 5. The first three sections introduce the datasets, evaluation metrics, and baselines that the models will be evaluated on. The fourth and fifth sections provide detailed information on how to reproduce the results and training, respectively. The last section describes the most important implementation details and dependencies.

4.1. Datasets and Benchmarks

MS MARCO Passage Ranking [62]. The Microsoft MAchine Reading COmprehension (MS MARCO) corpora are very popular benchmarks for information retrieval (IR) and web information systems (WIS) tasks. It is among the largest relevance datasets ever, with enough data to sufficiently train a ranking model for web search. This dataset contains 8.8 million passages extracted from web documents and is designed specifically to rank passages on relevance to queries. It has one passage human-labeled as relevant on average for over one million queries. These labels are binary, and the data contains many false negatives where a relevant passage is not labeled as such. The MS MARCO dataset is used for everything in this research, including training, validation, indexing, and evaluation.

TREC DL Track [12]. The MS MARCO dataset is complemented by the text retrieval conference (TREC) deep learning (DL) track, organized by the U.S. National Institute for Standards and Technology (NIST) [14]. Following similar IR research and community guidelines [46], we focus on the TREC DL 2019 [10] and TREC DL 2020 [11] benchmarks for our evaluation. These contain sets of 43 and 54 queries respectively accompanied by 9k (query, passage) pairs which are graded on a 0-3 relevancy scale provided by NIST. These labels are richer in information than the binary labels in the general dataset, enabling evaluation for deep learning models. Additionally, we evaluate our results on the DL-hard benchmark [56] which is constructed from the 25 most challenging topics from the DL ’19 and ’20 and 27 additional queries with new sparse judgments.

4.2. Evaluation Metrics

Passage ranking is a thoroughly researched field with a large community and well-established standards for evaluation metrics [46, 62]. To ensure comparability of our results with existing studies, we adapt these standard metrics where applicable. This section outlines these metrics along with their mathematical details.

As introduced in chapter 2, the passage ranking task is to sort passages in a corpus according to their relevance to a query. The terms document and passage are used interchangeably. This **relevancy**

$rel(q, d)$ between a query-document pair is made quantifiable with the binary labels in Equation 4.1, commonly referred to as $qrels$. However, the TREC DL tracks [10, 11] apply more sophisticated labels on a four-point scale: perfectly relevant (3), highly relevant (2), related (1), and irrelevant (0). Related passages are on the same topic, but do not answer the question. To compute the binary judgments required for some measures, only labels 2 and 3 should be considered relevant. These minimum relevancy scores are adhered to throughout my experiments.

$$rel(q, d) = \begin{cases} 1 & \text{document } d \text{ is above the minimum relevancy threshold for query } q \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Two foundational performance metrics for information retrieval from a corpus are **Precision P** (Equation 4.2) and **Recall R**. Where R is a ranked list of passages, with (i, d) denoting that document d is ranked at position i. Both of these metrics assume binary $qrels$ and are often evaluated at a cutoff k, denoted as P@k, meaning that only passages up until k are considered. $k = 1000$ for first-stage retrieval in the MS MARCO passage re-ranking task. Precision increases as false positives decrease, while recall improves as false negatives decrease.

$$P@k(R, q) = \frac{\sum_{(i,d)\in R} rel(q, d)}{k} \tag{4.2}$$

Precision and recall are mostly important for first-stage retrieval, which we assume to be provided. Re-ranking is primarily concerned with ranking the most relevant passages before non-relevant ones. Therefore, we need metrics that highlight the order of the top-ranked passages more.

The primary evaluation metric adopted in our research is **normalized Discounted Cumulative Gain (nDCG)**, described in Equation 4.3. This metric stems from Discounted Cumulative Gain (DCG), which is is specifically intended to capture gradient relevancy labels. It measures user satisfaction by rewarding the most-relevant (gradient) results to appear first. nDCG compares the DCG from the actual ranking against an "ideal" ranking, in which the passages are perfectly sorted by decreasing relevance. More specifically, nDCG@10 is reported in the TREC DL '19 [10] and TREC DL '20 [11] guidelines, and also in our research.

$$\begin{aligned} nDCG@k(R, q) &= \tfrac{DCG@k(R,q)}{IDCG@k(R,q)} \\ DCG@k(R, q) &= \textstyle\sum_{(i,d)\in R, i\leq k} \tfrac{2^{rel(q,d)}-1}{log_2(i+1)} \end{aligned} \tag{4.3}$$

Our secondary metric is the simple and common **(Mean) Reciprocal Rank RR**, Equation 4.4. RR is computed as the inverse rank of the first binary relevant document. This metric enforces a relevant result to appear near the ranking top, which is crucial for user satisfaction in web search.

$$\begin{aligned} RR(R, q) &= \tfrac{1}{rank(R,q)} \\ rank(R, q) &= min(i \mid (i, d) \in R \wedge rel(q, d)) \end{aligned} \tag{4.4}$$

Our third measurement is **(Mean) Average Precision AP**, Equation 4.5. Precision is only calculated on one binary relevant (query, passage) pair. AP computes the average precision across each $qrel$ and more accurately measures that relevant passages are ranked before non-relevant ones. The official evaluation measures for DL-HARD dataset [56] exclude AP, and we follow this convention.

$$AP(R, q) = \frac{\sum_{(i,d)\in R} P@i(R, q) \cdot rel(q, d)}{\sum_{d\in C} rel(q, d)} \tag{4.5}$$

Finally, the re-ranking latency is reported in milliseconds on CPU.

4.3. Baselines

Our models are compared against the following baselines:

- Sparse retrievers rely on exact term matching between queries and documents. The first baseline is **BM25** [71], which applies term frequency and inverse document frequency directly to the queries and documents. The second baseline is **SPLADE++ SelfDistil** [19], which is a more efficient version of the sparse neural retriever SPLADE [18]. This model contextualizes the terms before retrieval.
- All re-ranking pipelines adopt BM25 with cutoff 1000 as first-stage retrieval, as described in the MS MARCO passage re-ranking task [62]. We consider **TCT-ColBERT** [48] as an efficient dual-encoder baseline in three settings: dense retrieval, re-ranking, and interpolated re-ranking. Our models were trained as distilled TCT-ColBERT models, making it an upper bound for the efficiency-performance trade-off of our results.
- The final baseline is a fine-tuned heterogeneous dual-encoder consisting of a BERT-based document encoder and the **AvgTokEmb** query-encoder [45], which is proposed as embedding-based transformer in the cited paper. This query encoder is very efficient, leaving out self-attention completely. It mainly provides interesting insights on achievable efficiency and is partly integrated in our model designs.

4.4. Evaluation Details

Evaluation will be performed on the DelftBlue supercomputer from TU Delft [1], using 10 cores of the Intel XEON E5-6248R 24C 3.0GHz CPU. Enough RAM is required to store the indexes. The experiments are performed on TREC DL '19 (judged), TREC DL '20 (judged), and TREC DL-hard test datasets introduced in section 4.1. The query-encoders use batch sizes of 32. In the default passage re-ranking task [62, 46], a candidate top 1000 passages as first retrieved by BM25, and then re-ranked by relevance.

The latency results are averaged over multiple iterations, ignoring the first run because caches may not have been properly warmed up yet. Any pre-processing is ignored. This includes creating the document embedding indexes, loading it into memory, and creating the lexical ranking from first-stage retrieval. Experiments on query-encoding latency report the entire process from a batch of text queries to their query-encodings. This includes tokenization and query-encoding. It also includes document retrieval for `AvgEmb` models. Experiments on full end-to-end re-ranking latency include: tokenization, query-encoding, in-memory retrieval of document vectors, computing their relevancy scores as dot-products (Equation 2.8), score interpolation (Equation 2.2), and sorting.

All pre-trained encoders originate from the HuggingFace Hub [86]. The Pyserini toolkit [47] provides most of the corresponding indexes for the sparse and dense retrieval experiments in Table 5.1 and Table 5.2. The BM25 index is provided by PyTerrier [55]. PyTerrier also provides the framework for our hyperparameter tuning and re-ranking experiments. These tools are described in more detail in section 4.6.

The interpolation hyperparameter α is fine-tuned on MAP using the same 512 samples of the MS-MARCO development set. For the baselines, we set $\alpha = 0.03$ for TCT-ColBERT, $\alpha = 0.11$ for Avg-TokEmb. Our models use $\alpha = 0.09$ for $\text{AvgEmb}_{\text{10-docs}}$, $\alpha = 0.39$ for $\text{AvgEmb}_{\text{10-docs}}$ + AvgTokEmb, and $\alpha = 0.02$ for $\text{AvgEmb}_{\text{q,10-docs}}$.

4.5. Training details

The various `AvgEmb` models are trained using knowledge distillation with TCT-ColBERT as its teacher model. Training is performed on all training queries from the MSMARCO passage corpus [62], accesed via ir_datasets [54]. Each training instance consists of a single query and its teacher's encoding. These (query, encoding) input pairs are precomputed and stored. The training data is split to adhere to stan-

dard training etiquette [26], ensuring no overlap between training and validation data, thus preventing the model from memorizing the ”test answers”.

The loss function is the MSE loss (Equation 3.6) between the student and teacher embeddings. We use an Adam optimizer [39] with a learning rate of 0.001. Each validation is performed on the same 1000 sampled queries from the MSMARCO development set. The model is trained on a maximum of 50 epochs over the entire training dataset. Early stopping is implemented after 3 consecutive validation rounds without any improvement, which typically occurs within 2 epochs. The resulting models are evaluated on the datasets from section 4.1.

Our models are trained on the DelftBlue supercomputer from TU Delft [1], using two NVIDIA A100 Tensor Core GPUs. The smaller models with less than 10 documents were trained on 10 cores of the Intel XEON E5-6248R 24C 3.0GHz CPU. The training duration currently scales linearly with the amount of documents and becomes infeasible on a CPU for models with more documents.

Our models and training pipeline were implemented using PyTorch [66], PyTorch-Lightning [17], and Huggingface’s Transformers [86]. These are described in more detail in section 4.6.

4.6. Implementation Details

First-stage lexical retrieval involves transforming the queries using the BM25 ”terrier_stemmed” index, hosted by PyTerrier [55]). This index uses Terrier’s default Porter stemming [68] and removes stopwords from the query. This model provides the lexical rankings at cutoff 1000 for our re-ranking stage. The MS MARCO development set is used to determine the optimal values for the α hyperparameter for score interpolation.

Our model applies a dual-encoder architecture, with BERT-based document encoder and `AvgEmb` estimator as query encoder. This query-estimator maps queries to arbitrary 768-dimensional vector representations, similar to a regular query-encoder. TCT-ColBERT also serves as our teacher model for distillation training, providing the target query embeddings. The code for training and inference is made available [6]. Additionally, an initial setup for the dual-encoder training described in subsection 3.3.1 is made available as well.[1]

Many useful tools and libraries have been published by related research before. The main dependencies of this thesis are listed below.

- **Fast-Forward Indexes** [45]: an implementation of interpolated dual-encoder re-ranking. Our code relied heavily on this library, and extended it to include our estimator and various scripts for training, re-ranking inference, pre-computation, and data plotting.
- **PyTerrier** [55]: Python package for retrieval, hyperparameter tuning, and evaluation experiments. PyTerrier also hosts some indexes, such as BM25 variants for MS MARCO passage.
- **HuggingFace Transformers** [86]: Popular and widely adopted hub for pretrained neural models. This library provided us with any pretrained models, including BERT [15], TCT-ColBERT [48], and SPLADE++ SelfDistil [19].
- **PyTorch Lightning** [17]: PyTorch [66] is a popular framework for AI model creation and training. PyTorch Lightning makes PyTorch code more readable, scalable, reproducible, and easier to apply. We use these libraries to create and train our models.
- **ir_datasets** [54]: Provides a common interface to many IR ranking datasets. Used to easily access the MS MARCO and related TREC DL datasets.
- **Pyserini** [47]: toolkit for reproducible information retrieval research, used for our sparse retrieval and dense retrieval baselines.

[1] Training AvgEmb as dual-encoder, forked from J. Leonhardt, et al. [45]: https://github.com/bovdberg/dual-encoders

5

Results

This chapter presents the findings of this study, and is structured to address the research questions outlined in chapter 1. Each section corresponds to one or more research questions, presenting experiments in a visually appealing manner, followed by a detailed analysis of those results. The evaluation setup for these experiments is detailed in chapter 4.

This chapter commences with section 5.1, which presents the primary experiments related to the AvgEmb model architecture. section 5.2 then explores various model settings and alternative methods, including the number of documents and the distribution of document weights.

5.1. General Architecture

RQ1. Is it possible to achieve an efficient and accurate estimation of a query embedding in neural re-ranking by leveraging the lexically most relevant document embeddings?

RQ2. How can this approach be extended with semantic query information to improve performance without significantly compromising efficiency?

As outlined in chapter 3, our final model architecture addresses both initial research questions simultaneously. The main (initial) results present both approaches independently, though the latter results focus solely on the final model design to avoid redundant information and experiments.

5.1.1. Overall Performance and Efficiency

This subsection examines the two primary interesting aspects: the re-ranking latency (efficiency) and accuracy (model performance). These two metrics often negatively correlate and present a trade-off; hence why they are discussed together. The re-ranking efficiency and performance of the baseline models and our models are detailed in Table 5.1 and Table 5.2, respectively.

The AvgEmb estimator models are trained as distilled versions of TCT-ColBERT, as described in section 3.2. Therefore, the performance of this teacher model represents the upper limit (ceiling) for the AvgEmb performance. The results demonstrate that the $\text{AvgEmb}_{\text{q,10-docs}}$ estimator, referred to as AvgEmb, indeed achieves performance comparable ($99.4\%, 97.6\%$ nDCG_{10}) to its teacher model on the first two datasets while reaching a speed-up of 13.4X. However, there are slightly bigger differences (93.0%) on the other datasets. Similar differences also exist in the comparison between the TCT-ColBERT and AvgTokEmb baselines on which the model is based. Our model outperforms the AvgTokEmb on all metrics and datasets except RR on TREC '19, indicating that it is more powerful in capturing the semantics than averaging over tokens alone.

	TREC-DL-Psg '19			TREC-DL-Psg '20			TREC-DL-Hard	
	$nDCG_{10}$	RR	AP	$nDCG_{10}$	RR	AP	$nDCG_{10}$	RR
Sparse Retrieval:								
BM25	0.480	0.642	0.286	0.494	0.619	0.293	0.274	0.422
SPLADE++ CC-SD	0.736	0.901	0.5	0.728	0.836	0.514	-	-
Dense Retrieval:								
TCT-ColBERT	0.670	0.823	0.391	0.668	0.815	0.284	-	-
Re-Ranking (no int.):								
BM25	0.480	0.642	0.286	0.494	0.619	0.293	0.274	0.422
↳TCT-ColBERT	0.679	0.828	0.430	0.676	0.822	0.452	0.369	0.534
Interpolated Re-Ranking:								
BM25	0.480	0.642	0.286	0.494	0.619	0.293	0.274	0.422
↳TCT-ColBERT	0.694	0.822	0.438	0.695	0.832	0.465	0.385	0.537
↳AvgTokEmb	0.677	0.884	0.404	0.610	0.752	0.394	0.337	0.492
↳$\text{AvgEmb}_{\text{10-docs}}$	0.571†	0.716	0.382†	0.581†	0.677†	0.388†	0.319	0.483
↳AvgTokEmb	0.694	0.852	0.428	0.641†	0.806	0.421†	0.346	0.484
↳$\text{AvgEmb}_{\text{q,10-docs}}$	0.690	0.832	0.439	0.679	0.815	0.449	0.356	0.522

Table 5.1: Re-ranking performance measured on different MSM-psg testsets at retrieval depth $k_s = 1000$. AP and RR use a minimum relevance of 2. † indicates that the model performs significantly worse on the metric than the interpolated re-ranking TCT-ColBERT baseline, as determined by a paired t-test ($p \leq 0.05$) [16].

	Query-Encoding				Full Re-ranking		
	Total 128q	Query 1q	Batch 32q	Speedup X	Q-Encoding %	Total 128q	Speedup X
Interpolated Re-Ranking:							
BM25							
↳TCT-ColBERT	1021	7.97	31.89	1	39.12	2609	1
↳AvgTokEmb	7.09	0.06	0.22	143.9	0.43	1634	1.6
↳$\text{AvgEmb}_{\text{10-docs}}$	64.8	0.51	2.03	15.8	3.96	1636	1.6
↳AvgTokEmb	68.59	0.54	2.14	14.9	2.05	3350	0.8
↳$\text{AvgEmb}_{\text{q,10-docs}}$	75.98	0.59	2.37	13.4	4.55	1671	1.6

Table 5.2: Latency measured in milliseconds on 128 sampled queries of MSM-Psg dev set [62], using batch size 32. Speedup is compared against TCT-ColBERT re-ranking.

We first evaluate the performance table Table 5.1. The SPLADE++ CoCondenser-SelfDistil [19] learned sparse model baseline reaches the highest performance on all measurements. However, this model was not created for efficiency. Although research on the SPLADE model variants has since been extended by an efficiency study [41].

Moving on, the interpolated re-ranking TCT-ColBERT baseline takes second place on almost all measurements. This was expected to outperform our models, as this is their teacher.

Continuing on to our models, the $\text{AvgEmb}_{\text{10-docs}}$ model alone lacks semantical insight and does not achieve competitive performance with only 0.571 nDCG_{10}. The highest performance on the first dataset is instead reached by the combinatorial pipeline of "BM25 + $\text{AvgEmb}_{\text{docs}}$ + AvgTokEmb", essentially computing the similarity score ϕ as:

$$\phi^{Combo}(q,d) = \alpha_1 \cdot \phi_S(q,d) + \alpha_2 \cdot \phi_D^{AvgEmb_{docs}}(q,d) + (1-\alpha_1-\alpha_2) \cdot \phi_D^{AvgTokEmb}(q,d), \tag{5.1}$$

where score interpolation hyperparameter α is split into three parts, such that $0 \leq \alpha_1 + \alpha_2 \leq 1$.

This pipeline outperforms both of its individual components $\text{AvgEmb}_{\text{10-docs}}$ and AvgTokEmb on almost all metrics. It even exceeds the interpolated TCT-ColBERT baseline on TREC '19 reciprocal rank (RR), possibly because the AvgTokEmb is trained and applied independently. It does not learn to simulate TCT-ColBERT encodings, but it is trained on re-ranking loss independently. Its performance across the different metrics relies partly on TCT-ColBERT and partly on AvgTokEmb. Though this pipeline still performs significantly worse on nDCG_{10} and AP on TREC DL '20. Although seemingly promising, it must be stressed that this approach requires two indexes, doubling the re-ranking time as dense scores must be calculated independently on either index and combined. This shortcoming was also addressed in chapter 3. However, the performance of this setup partly served as an intermediate motivation to integrate AvgTokEmb in our AvgEmb model design.

Most importantly, these results show that our final $\text{AvgEmb}_{\text{q,10-docs}}$ model does not perform significantly worse (†) [16] than its teacher model on any measurement. It also outperforms the combinatorial pipeline described above on all metrics across the TREC DL '20 and DL-Hard datasets.

We continue by analyzing the efficiency in Table 5.2. It becomes immediately clear that the baseline AvgTokEmb encoder remains the fastest solution, which is expected as it is completely integrated in the AvgEmb estimator.

It may seem odd that there exists a relatively big difference between the query encoding latency between these models. This difference is further analyzed and explained in subsection 5.1.2. The full re-ranking pipeline shows that query encoding is reduced from 39.12% of the total runtime to less than 4.55% in all proposed models. This results in a 1.6X speed-up compared to TCT-ColBERT in all cases with one index.

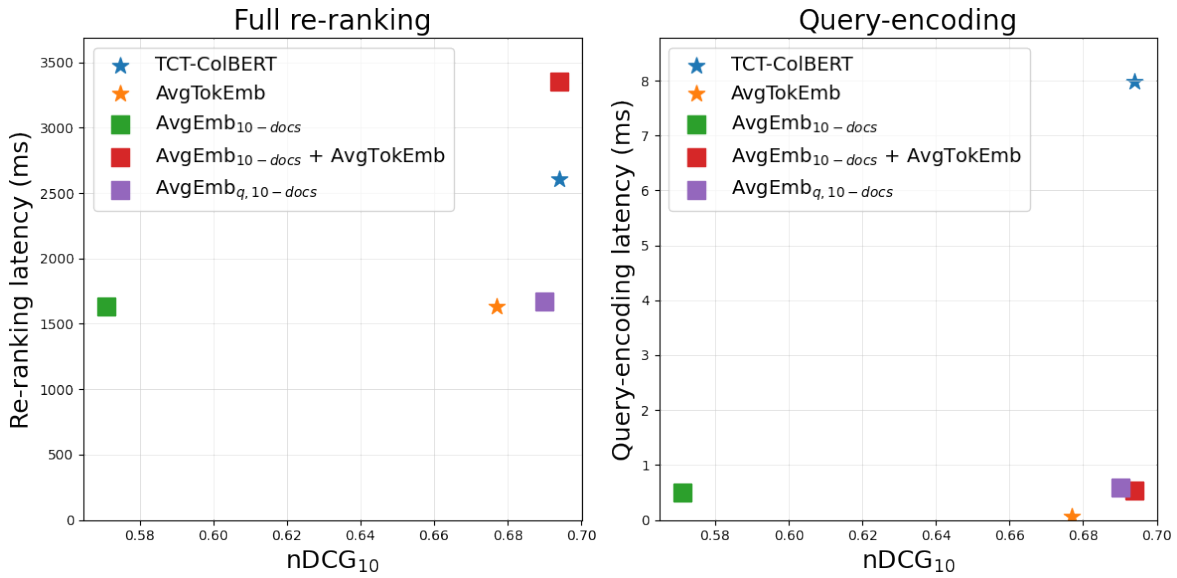


Figure 5.1: Re-ranking latency distribution measured in milliseconds per query on 128 sampled queries of MSM-Psg dev set [62], using batch size 32. First graph shows full re-ranking runtime, second focuses on the query encoding latency per query. Performance is measured on TREC-DL-Psg '19.

The trade-off space between latency and performance is illustrated more clearly in Figure 5.1. The left graph relates to the full re-ranking latency, while the right graph focuses on query encoding latency. The query encoding latency is addressed independently because this research leaves all variables except the query encoding unchanged and constant.

The key insights from these graphs are: 1) The differences between AvgEmb and TCT-ColBERT suggest that mayor efficiency improvements are made with only marginal sacrificed performance. 2) The

vertical difference between AvgEmb and AvgTokEmb indicates that while our query-encoding is not as efficient as this extremely lightweight approach, the latency differences when comparing the full re-ranking runtime are negligible.

Re-ranking Latency Distribution

The distribution of the re-ranking latency is depicted in Figure 5.2 in a similar fashion to Figure 1.1, the latency is segmented vertically into its three primary components: query encoding, retrieval of document vectors from the index, and similarity score computation. Any remaining latency is categorized as *other*.

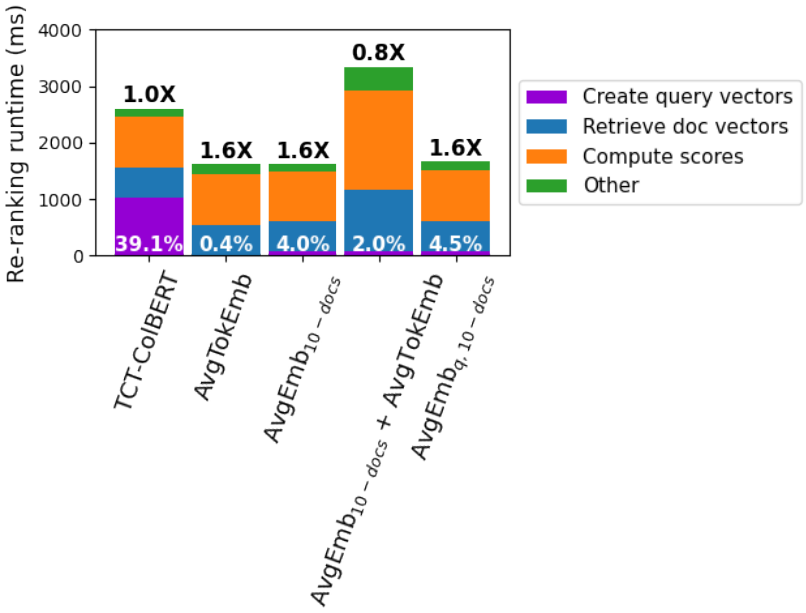


Figure 5.2: Re-ranking latency distribution measured in milliseconds per query on 128 sampled queries of MSMARCO-Passage development set [62], using batch size 32. Speedup against TCT-ColBERT re-ranking is shown above each bar. Query encoding percentage is shown below.

This plot shows that only the query encoding latency varies between models, while other operations remain constant in latency. This is expected, as the focus of this research is specifically on reducing re-ranking latency by improving query encoding latency. The exception to the above is the $AvgEmb_{docs} + AvgtokEmb$ pipeline, which consists of two re-ranking pipelines, each with its own index and associated operations, thereby doubling the latency for index-related operations.

An ambiguity in this graph concerns the retrieval of top-ranked document vectors for AvgEmb pipelines, which can be included under either ”Retrieve doc vectors” or ”Encode queries”. We chose to include it in the query encoding latency because these vectors are required for query estimation. This will be further be discussed in subsection 5.1.2.

5.1.2. Query Encoding Latency

The in-depth query-encoding latency distribution for each part of the new AvgEmb Estimator architecture is visualized in Figure 5.3. The latency is again vertically divided into primary components: 1) Creating the lightweight semantic AvgTokEmb query-encoding q_{light} as the weighted average of the token embeddings. 2) Retrieving the embeddings of the n_docs top-ranked documents from the index. 3) Estimating the query vector $\hat{q}$ as the weighted average of its embeddings.

In theory, AvgEmb should be able to reach a much higher efficiency. The top-ranked document embeddings are a requirement to calculate the dense scores, so they are also already retrieved outside the

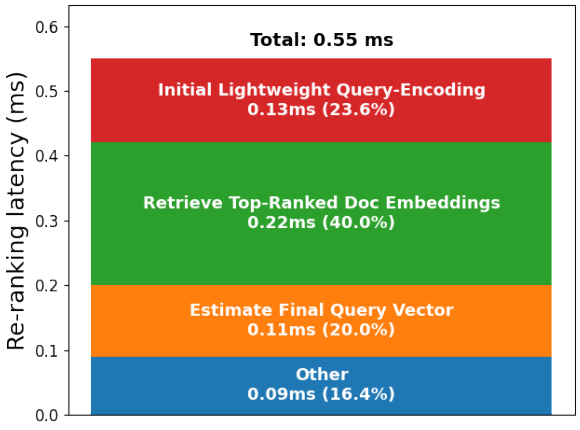


Figure 5.3: Query-encoding latency distribution of AvgEmb Estimator, measured in milliseconds per query on 128 sampled queries of MSM-Psg dev set [62], using batch size 32.

query encoding. If these vectors could be passed directly to the query encoder on initialization, then this step becomes obsolete, reducing the query encoding latency by a major 40%.

A second observation is that Table 5.2 shows that AvgTokEmb can encode 128 queries in just 7.09 milliseconds. Both query-encoding steps should each be able to reach similar efficiency as this encoder, since they essentially apply the same weighted average computation. This could potentially improve the encoding with another 18.14%, resulting in just $7.09 + 7.09 + 0.09 = 14.27$ milliseconds total. This would make query estimation merely 0.89% of re-ranking time. This speculative query-encoding reaches a speed-up of 72.5X over the TCT-ColBERT baseline, remaining at a total of 1.6X re-ranking speed-up.

5.1.3. Correlation to Lexical Performance

We hypothesize that the algorithm's performance in part relies on BM25 performance, as the top-ranked documents originate directly from the sparse ranking. This should be especially evident for our $\text{AvgEmb}_{\text{docs}}$ variant which disregards the semantics of the q_light lightweight query encoding and solely considers the lexical document weights. To this end, Figure 5.4 highlights the correlation between NDG_{10} performances of BM25, $\text{AvgEmb}_{\text{10-docs}}$, and $\text{AvgEmb}_{\text{q,10-docs}}$.

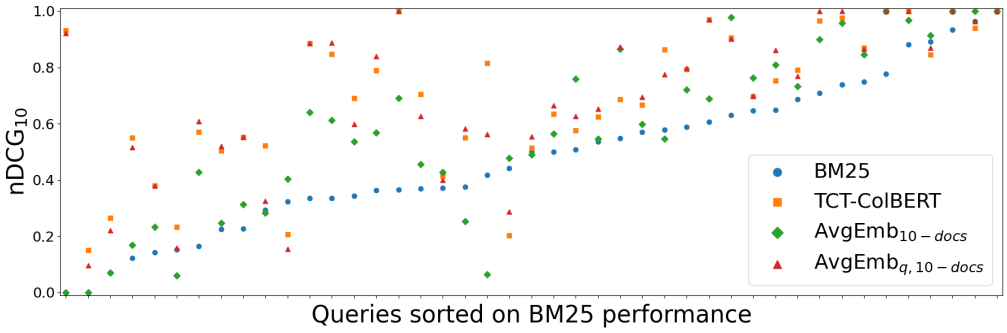


Figure 5.4: nDCG_{10} performances of the models on queries from the MSM-Psg TREC 2019 dataset, sorted on increasing BM25 performance.

The data confirms that the model without query encoding indeed depends on the BM25 performance. This dependence on the BM25 ranking is the clearest in the first three examples, where their performance is equivalent. The full model correlates more with the TCT-ColBERT performance than with BM25. In almost all cases, the performance fluctuates between its lexical and semantical parts, which aligns with the theoretical given the nature of the model architecture and training.

5.2. Exploration of Alternatives

RQ3. What alternative approaches and settings could further boost the model performance or efficiency?

This section provides an ablation study on various model variable settings that have been assumed as defaults in earlier chapters. These experiments are conducted to investigate some alternatives to these variables and approaches.

In subsection 3.1.1, the AvgEmb model design raised two topics that set the grounds for further exploration: the optimal number of documents to average over, and how the embedding weights should be distributed. These topics are analyzed in subsection 5.2.1 and subsection 5.2.2, respectively.

5.2.1. Number of Documents

"What is the optimal amount of documents to average over?"

Impact on Weight Distributions

Figure 5.5 presents the learned document weight distributions across different AvgEmb models with different n_docs settings. Each mark on the X-axis represents the weight at the corresponding lexical rank.

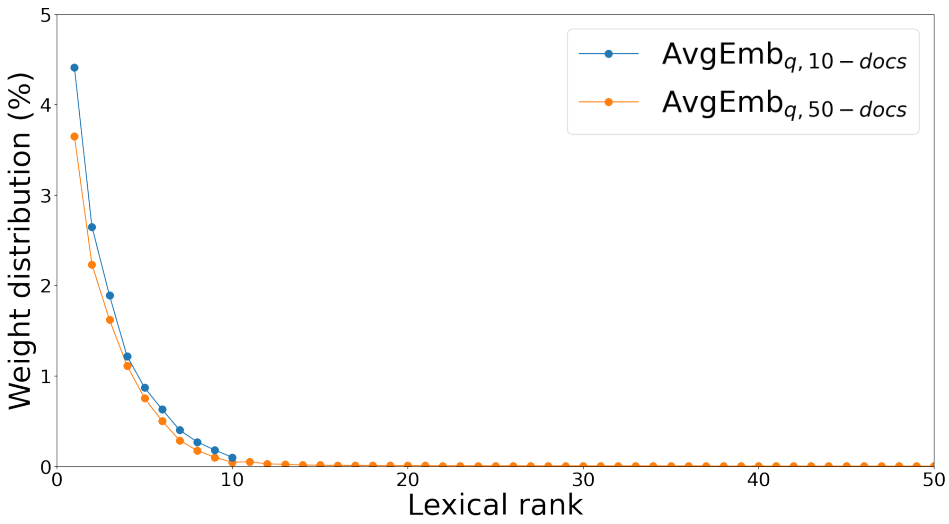


Figure 5.5: Learned embeddings weights at each lexical rank for different number of documents n); q_{light} is excluded.

The model learns that the q_{light} encoding alone is already an appropriate estimation, allocating it a significant portion of the total weight, ranging from 0.83 to 0.86 across all models. However, this weight is excluded from the graph to focus on the document embedding weights.

The remaining weight is distributed among the document embeddings. The weight assigned to each document at rank d_i decreases proportionally to an increase of n_docs as expected, because a similar amount of remaining weight has to be distributed among more vectors.

The graph reveals a negative correlation between the weights and their lexical document ranks d_i, which highlights the BM25 model's effectiveness in ranking the documents according to their relevance.

Another interesting observation is that the AvgEmb$_{50\text{-docs}}$ model demonstrates that the learned document weights approach zero after rank 10. Embeddings with near-zero weight will barely impact the final query estimation, suggesting that 10 or fewer document embeddings might be sufficient.

Impact on Performance and Efficiency

To further substantiate this claim, it is important to revisit the performance-efficiency trade-off and demonstrate how models with varying numbers of documents perform. This analysis is represented in Figure 5.6, structured similarly to Figure 5.1 and incorporating the same baselines.

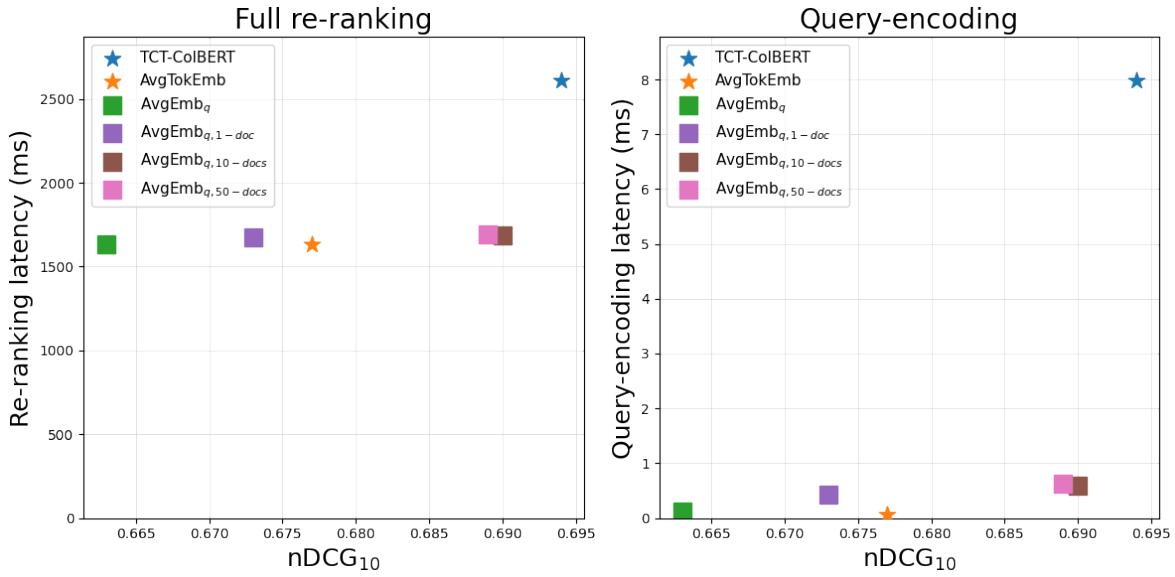


Figure 5.6: Comparing performance against efficiency using different n_docs against baselines TCT-ColBERT re-ranking and AvgTokEmb. Latency is measured in milliseconds per query on 128 sampled queries of MSM-Psg dev set [62], using batch size 32. First graph shows full re-ranking runtime, second focuses on the query encoding latency per query.

Firstly, it is noteworthy that the horizontal performance scale displays little variation, ranging from 0.663 (q_only) to 0.695 (TCT-ColBERT). This indicates that the majority of models achieve comparable performance levels.

Even though the q_only model only considers the token averaging part of the model, it demonstrates lower performance than AvgTokEmb. The reasoning behind this is that it was trained via distillation with TCT-ColBERT, whereas AvgTokEmb was trained as an independent dual-encoder setting on ranking loss.

The estimations become more accurate by integrating the document embeddings. This increase becomes most evident when comparing the models q_only, $n\_docs = 1$, and $n\_docs = 10$. The performances increase linearly from 0.663 to 0.673 to 0.690.

With the improvements suggested in subsection 5.1.2, the document embeddings can be reused throughout re-ranking without any additional latency. However, it is not advisable to set n_docs equal to the sparse cutoff of 1000, as embeddings at lower ranks (even with near-zero weights) may negatively impact the final query estimation. This is marginally evidenced by AvgEmb$_{50\text{-docs}}$ achieving slightly inferior performance compared to the model trained on 10 documents. Given that the model with 10 trained documents marginally outperforms the other configurations, we adopt this as the default setting.

5.2.2. Alternative Weighting Methods

"How should the embedding weights be distributed?"

This final question concerns the weight distribution of the learned models, aiming to determine whether these weights align closely with a standardized weight distribution method. Such approximated weights might be even more optimal given our somewhat trivial training setup, improving the performance without compromising efficiency. A standardized distribution could improve the results by slightly correcting

the learned values; for instance, the learned weight for d_{11} has a higher weight than d_{10} in AvgEmb$_{50\text{-docs}}$ which would be corrected.

If a particular method shows perfect alignment with our data, it could instigate a pivoted version of our model design. In this revised design, a hyperparameter ranging from 0 to 1 would determine the weight of the lightweight query encoding q_{light}, while the remaining weight would be distributed over the document ranks according to the selected standard weight distribution.

Figure 5.7 compares the learned weights $W_{d\_embs} : \{\omega_i \mid i \in \text{lexical ranks}\}$ against various probability distributions. These weights act as a baseline measurement in this experiment (blue — line).

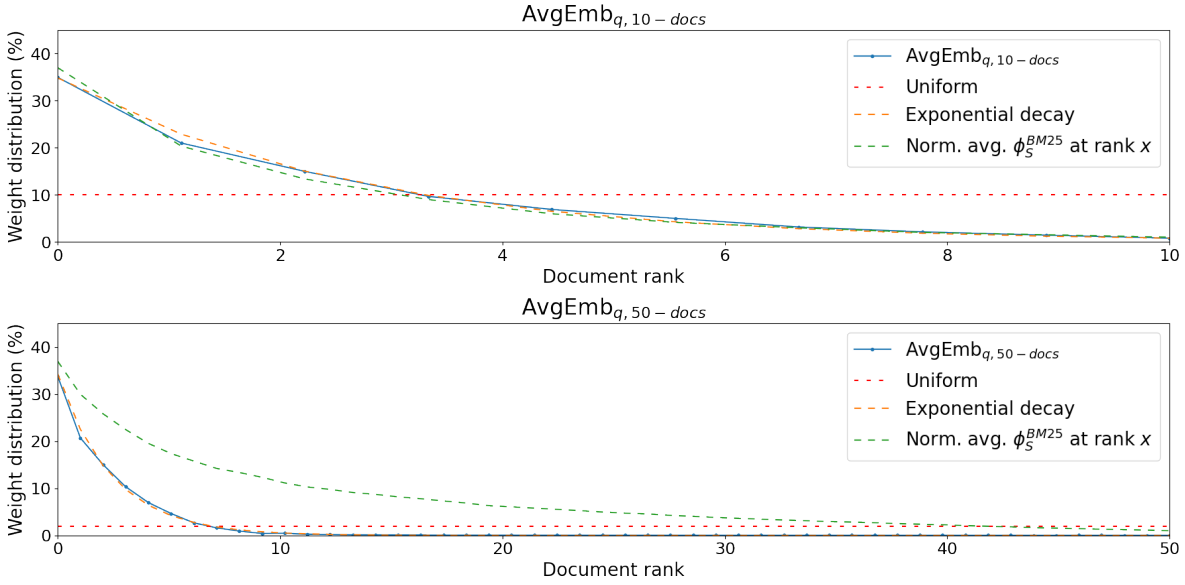


Figure 5.7: Comparison of different weight distribution methods for 10 documents, ignoring the query encoding weight. Softmax scores is calculated from average score at rank d_i for 1024 validation queries. The first graph shows weight approximations for n_docs=10, the second for n_docs=50.

A trivial method is uniform distribution, assigning equal weights to all embeddings (red - - line):

$$\omega_i^{uniform} = \frac{1}{k_S}, \tag{5.2}$$

where k_S represents the retrieval depth.

This uniform method clearly does not match the learned weights. Instead, the decrease in document weights appears to decay exponentially, with lesser decreases at higher ranks. Consequently, we introduce an exponential decay function (orange - - line):

$$\omega_i^{exp\_decay} = 0.52e^{-0.42 \cdot i}, \tag{5.3}$$

The constants were gathered through trial and error. This line indeed closely approximates the learned weights, suggesting it as a viable option for AvgEmbq,10-docs. We introduce a third method that involves the sparse scores themselves by normalizing them and rescaling (green - - line):

$$\omega_i^{BM25} = scale(\ norm(\ avg(\phi_s^{BM25}, i\ ))) \tag{5.4}$$

In the upper graph, this version also closely resembles the baseline. However, the bottom graph indicates that this approach does not hold up when applied to the variant with 50 top documents. Nonetheless, reaching the same conclusion as in subsection 5.2.1, designing the weights to appropriate 10 documents again seems to be sufficient.

6

Conclusion

Efficiency and accuracy are critical factors in the domain of information retrieval and search engines. Our research is focused on efficiency in the task of passage re-ranking. In this thesis, we consider a dual-encoder architecture with representation learning and score interpolation. Dual-encoders are known for their inference efficiency because they allow their document embeddings to be pre-computed and indexed. The first stage employs the classical sparse retrieval method BM25 [72] at retrieval depth $k_S = 1000$ (Equation 2.4), resulting in lexical scores $\phi_S(q, d)$. This is followed by a re-ranking step consisting of: creating query representations, retrieving k_S passage representations for each query, computing similarity scores $\phi_D(q, d)$ for all query-passage representation pairs as their dot products (Equation 2.8), interpolating the sparse and dense scores (Equation 2.2) and re-ranking them according to their interpolated scores.

In this chapter, the research questions are repeated and answered in linear order. We then discuss the limitations of this research and propose several ideas for possible improvements.

RQ1: "Is it possible to achieve an efficient and accurate estimation of a query embedding in neural re-ranking by leveraging the lexically most relevant document embeddings?"
In response to RQ1, we propose a novel method to estimate an accurate query vector representation. This approach is characterized by its efficiency; it relies solely on retrieving document embeddings from an in-memory index and simple mathematical operations. Our initial proposed $\text{AvgEmb}_{\text{n-docs}}$ query estimator model (Equation 3.1) leverages the embeddings of the query's n top-ranked passages from first-stage retrieval, assigning weights according to their lexical ranks, and transforming them into a 1-dimensional vector representation via weighted averaging over the document embeddings.

We show that $n = 10$ is the most effective setting, although it performs significantly worse compared to its teacher model TCT-ColBERT in the same setting. We conclude that this approach is very efficient but does not achieve competitive performance on its own. Most likely because it does not take semantic information into account, leaving it vulnerable to the vocabulary mismatch problem [21].

RQ2: "How can this approach be extended with semantic query information to improve performance without significantly compromising efficiency?"
The second research question is specifically aimed at mitigating this problem. To address RQ2, we extend our estimator with an additional vector q_{light} that is generated by a lightweight query encoding approach. We refer to this new model as $\text{AvgEmb}_{\text{q,n-docs}}$ (Figure 3.4, Equation 3.4, Equation 3.5), abbreviated to just AvgEmb.

To compute q_{light}, we integrate an efficient query encoder AvgTokEmb, proposed by Leonhardt et al. [45] as an embedding-based encoder. This encoder operates by retrieving query token embeddings e_t from a token embedding matrix E_T and computing their average representation. E_T is initialized from the BERT embedding layer [15]. We extend this encoder with learned weights for each token in the vocabulary, suggesting that this would assign lower weights to prevalent general tokens such as stop words and higher weights to context-defining tokens. This q_{light} is simply concatenated to the n document embeddings and assigned its own learned weight. Since q_{light} already estimates the query embedding somewhat accurately, it is assigned a substantially larger weight than the documents.

The resulting AvgEmb model, visualized in Figure 3.4, still relies on only efficient operations (Equation 3.4): retrieving token embeddings from a matrix, retrieving document embeddings from an in-memory index, and computing weighted averages. Moreover, our model is trained as a distilled version of TCT-ColBERT [48] using MSE loss [82]. In this training setup, the teacher's performance serves as an upper boundary on the student's performance.

We argue that $n = 10$ is again the most effective setting. The AvgTok$_{\text{q,10-docs}}$ estimator achieves a query encoding speed-up of 13.4X over its TCT-ColBERT teacher while retaining 98.6% of its performance on the TREC-DL-Psg '19 and '20 datasets [10, 11] and 93.0% on more complex queries from DL-HARD [56]. Overall, this results in a 1.6X efficiency gain in the full interpolated passage re-ranking pipeline on CPU.

The current query estimation latency distribution is elucidated in Figure 5.3. This graph shows that 40% of query-encoding is spent on retrieving the document embeddings. These same document embeddings are retrieved for the final dot product computation. These embeddings could be shared between the estimator and similarity score computation, greatly improving efficiency without any performance loss. Additionally, this transformation would ensure that increasing the number of top-ranked documents does not result in a significant delay.

RQ3: "What alternative approaches and settings could further boost the model performance or efficiency?"
Regarding the third research question, we first consider how many top-ranked documents should be used for the best performance. Figure 5.6 highlights that AvgEmb$_{\text{q,10-docs}}$ model marginally outperforms other configurations. We therefore employ $n = 10$ as the default setting as seen above.

Figure 5.5 showcases that the AvgEmb$_{\text{50-docs}}$ model learns to assign near-zero weights to documents at rank $d_i > 10$. Consequently, these document embeddings barely impact the final representations. We argue that these documents at higher ranks would otherwise incur noise, making the final query estimation prediction less accurate.

Secondly, we evaluate the learned weights against different weight distribution methods. We are able to closely approximate the learned weights of AvgEmb$_{\text{q,10-docs}}$ with an exponential decay function. This function could be used to further smooth out the learned weights, though we have not yet explored this.

Future Directions

Regarding efficiency, as speculated above in RQ2, re-using the document representations between the query estimator and the similarity score computation could result in a 40% decrease in query latency.

Secondly, the validity of our approach could be further increased by evaluating on different datasets and scenarios. Some suggestions include the BeIR benchmark [80] that contains various datasets for zero-shot evaluation, Google's Natural Questions (NQ) dataset [40] for open-domain question answering (QA) to test longer complex queries, and applying the estimator directly in dense passage retrieval.

The following future research suggestions revolve around improving prediction accuracy. In RQ3 above, we propose to smooth out the learned weights with an exponential decay function approximation. Furthermore, the main limitation of our training setup is that the teacher model performance serves as an

upper bound for student effectiveness. We explore ideas for more complex training in subsection 3.3.1. These include a dual-encoder setup [48, 45], mining hard negatives with TAS-B [30], and an ensemble of teachers [31].

Finally, our method might also be extended by retrieval augmented generation (RAG) [24]. One of my peers is exploring currently exploring the possibility to include RAG in the Fast-forward setting [41] we also employed.

Acknowledgments

The current training setup, while seemingly simple, marks a significant personal milestone in my journey with AI model training. Despite having no prior experience in this domain, I successfully developed a functional, efficient, and effective framework.

First and foremost, I would like to express my deepest gratitude to my supervisors Jurek Leonhardt and Avishek Anand for their continuous support, guidance, and encouragement throughout my research. Their expertise and insight have been invaluable in shaping this work. I am also grateful to TU Delft for providing the necessary resources for my research. I have thoroughly enjoyed the active and encouraging Web Information Systems research group for their information sharing, presentations, discussions, and feedback. Lastly, a special thanks to my family and friends for their unwavering support, comfort, and patience throughout this journey.

In full transparency, much text in this thesis has been rephrased using various AI tools powered by ChatGPT, these generated texts have been checked for validity. I am obliged to disclose this according to TUD policies.[1]

[1] TU Delft publishing policies: https://www.tudelft.nl/library/actuele-themas/open-publishing/about/policies

References

[1] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 2)*. https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2. 2024.

[2] 3Blue1Brown. *Attention in Transformers, Step-by-Step | DL6*. Accessed: 2025-03-03. 2023. URL: https://www.youtube.com/watch?v=eMlx5fFNoYc.

[3] 3Blue1Brown. *Dot products and duality | Chapter 9, Essence of linear algebra*. https://www.youtube.com/watch?v=LyGKycYT2v0. Accessed: 2025-03-02. 2016.

[4] Shun-ichi Amari. "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* 5.4-5 (1993), pp. 185–196.

[5] Yang Bai et al. "SparTerm: Learning Term-based Sparse Representation for Fast Text Retrieval". In: *CoRR* abs/2010.00768 (2020). arXiv: 2010.00768. URL: https://arxiv.org/abs/2010.00768.

[6] Bo van den Berg. *AvgEmb GitHub repository: Efficient interpolation-based ranking on CPUs*. https://github.com/BovdBerg/fast-forward-indexes. 2025.

[7] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

[8] Sebastian Bruch, Siyu Gai, and Amir Ingber. "An Analysis of Fusion Functions for Hybrid Retrieval". In: *ACM Trans. Inf. Syst.* 42.1 (Aug. 2023). ISSN: 1046-8188. DOI: 10.1145/3596512. URL: https://doi.org/10.1145/3596512.

[9] Nachshon Cohen, Yaron Fairstein, and Guy Kushilevitz. "Extremely efficient online query encoding for dense retrieval". In: *NAACL 2024*. 2024. URL: https://www.amazon.science/publications/extremely-efficient-online-query-encoding-for-dense-retrieval.

[10] Nick Craswell et al. "Overview of the TREC 2019 deep learning track". In: *arXiv preprint arXiv:2003.07820* (2020).

[11] Nick Craswell et al. "Overview of the TREC 2020 deep learning track". In: *CoRR* abs/2102.07662 (2021). arXiv: 2102.07662. URL: https://arxiv.org/abs/2102.07662.

[12] Nick Craswell et al. "TREC Deep Learning Track: Reusable Test Collections in the Large Data Regime". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. <conf-loc>, <city>Virtual Event</city>, <country>Canada</country>, </conf-loc>: Association for Computing Machinery, 2021, pp. 2369–2375. ISBN: 9781450380379. DOI: 10.1145/3404835.3463249. URL: https://doi.org/10.1145/3404835.3463249.

[13] Zhuyun Dai and Jamie Callan. "Deeper Text Understanding for IR with Contextual Neural Language Modeling". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR'19. Paris, France: Association for Computing Machinery, 2019, pp. 985–988. ISBN: 9781450361729. DOI: 10.1145/3331184.3331303. URL: https://doi.org/10.1145/3331184.3331303.

[14] USA Department of Commerce. *National Institute of Standards and Technology*. URL: https://webbook.nist.gov/chemistry/.

[15] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[16] Rotem Dror et al. “The Hitchhiker‘s Guide to Testing Statistical Significance in Natural Language Processing”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Iryna Gurevych and Yusuke Miyao. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 1383–1392. DOI: 10.18653/v1/P18-1128. URL: https://aclanthology.org/P18-1128/.

[17] William Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Version 1.4. Mar. 2019. DOI: 10.5281/zenodo.3828935. URL: https://github.com/Lightning-AI/lightning.

[18] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. “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*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 2288–2292. ISBN: 9781450380379. URL: https://doi.org/10.1145/3404835.3463098.

[19] Thibault Formal et al. “From Distillation to Hard Negative Sampling: Making Sparse Neural IR Models More Effective”. In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’22. Madrid, Spain: Association for Computing Machinery, 2022, pp. 2353–2359. ISBN: 9781450387323. DOI: 10.1145/3477495.3531857. URL: https://doi.org/10.1145/3477495.3531857.

[20] Thibault Formal et al. *SPLADE v2: Sparse Lexical and Expansion Model for Information Retrieval*. 2021. DOI: 10.48550/ARXIV.2109.10086. URL: https://arxiv.org/abs/2109.10086.

[21] G. W. Furnas et al. “The vocabulary problem in human-system communication”. In: *Commun. ACM* 30.11 (Nov. 1987), pp. 964–971. ISSN: 0001-0782. DOI: 10.1145/32206.32212. URL: https://doi.org/10.1145/32206.32212.

[22] Prakhar Ganesh. “Knowledge Distillation: Simplified”. In: *Towards Data Science* (2019). Accessed: 2025-02-28. URL: https://towardsdatascience.com/knowledge-distillation-simplified-dd4973dbc764.

[23] Luyu Gao and Jamie Callan. “Unsupervised Corpus Aware Language Model Pre-training for Dense Passage Retrieval”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 2843–2853. DOI: 10.18653/v1/2022.acl-long.203. URL: https://aclanthology.org/2022.acl-long.203/.

[24] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL]. URL: https://arxiv.org/abs/2312.10997.

[25] Tiezheng Ge et al. “Optimized product quantization”. In: *IEEE transactions on pattern analysis and machine intelligence* 36.4 (2013), pp. 744–755.

[26] Varun Godbole et al. *Deep Learning Tuning Playbook*. Version 1.0. 2023. URL: http://github.com/google-research/tuning_playbook.

[27] Google. *Spam Policies for Google Web Search*. 2025. URL: https://developers.google.com/search/docs/essentials/spam-policies.

[28] Jianping Gou et al. “Knowledge distillation: A survey”. In: *International Journal of Computer Vision* 129.6 (2021), pp. 1789–1819.

[29] Zellig S Harris. “Distributional structure”. In: *Word* 10.2-3 (1954), pp. 146–162.

[30] Sebastian Hofstätter et al. “Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling”. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 113–122. ISBN: 9781450380379. DOI: 10.1145/3404835.3462891. URL: https://doi.org/10.1145/3404835.3462891.

[31] Sebastian Hofstätter et al. “Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation”. In: *CoRR* abs/2010.02666 (2020). arXiv: 2010.02666. URL: https://arxiv.org/abs/2010.02666.

[32] H Jabbar and Rafiqul Zaman Khan. “Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)”. In: *Computer science, communication and instrumentation devices* 70.10.3850 (2015), pp. 978–981.

[33] Herve Jegou, Matthijs Douze, and Cordelia Schmid. “Product quantization for nearest neighbor search”. In: *IEEE transactions on pattern analysis and machine intelligence* 33.1 (2010), pp. 117–128.

[34] Jeff Johnson, Matthijs Douze, and Hervé Jégou. “Billion-Scale Similarity Search with GPUs”. In: *IEEE Transactions on Big Data* 7.3 (2021), pp. 535–547. DOI: 10.1109/TBDATA.2019.2921572.

[35] Euna Jung, Jaekeol Choi, and Wonjong Rhee. “Semi-Siamese Bi-encoder Neural Ranking Model Using Lightweight Fine-Tuning”. In: *CoRR* abs/2110.14943 (2021). arXiv: 2110.14943. URL: https://arxiv.org/abs/2110.14943.

[36] Vladimir Karpukhin et al. “Dense Passage Retrieval for Open-Domain Question Answering”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber et al. Online: Association for Computational Linguistics, Nov. 2020, pp. 6769–6781. DOI: 10.18653/v1/2020.emnlp-main.550. URL: https://aclanthology.org/2020.emnlp-main.550.

[37] Omar Khattab and Matei Zaharia. “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. Virtual Event, China: Association for Computing Machinery, 2020, pp. 39–48. ISBN: 9781450380164. DOI: 10.1145/3397271.3401075. URL: https://doi.org/10.1145/3397271.3401075.

[38] Seungyeon Kim et al. “EmbedDistill: A geometric knowledge distillation for information retrieval”. In: *arXiv preprint arXiv:2301.12005* (2023).

[39] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[40] Tom Kwiatkowski et al. “Natural Questions: a Benchmark for Question Answering Research”. In: *Transactions of the Association of Computational Linguistics* (2019).

[41] Carlos Lassance and Stéphane Clinchant. “An Efficiency Study for SPLADE Models”. In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’22. Madrid, Spain: Association for Computing Machinery, 2022, pp. 2220–2226. ISBN: 9781450387323. DOI: 10.1145/3477495.3531833. URL: https://doi.org/10.1145/3477495.3531833.

[42] Carlos Lassance et al. *SPLADE-v3: New baselines for SPLADE*. 2024. arXiv: 2403.06789 [cs.IR]. URL: https://arxiv.org/abs/2403.06789.

[43] Victor Lavrenko and W. Bruce Croft. “Relevance based language models”. In: *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’01. New Orleans, Louisiana, USA: Association for Computing Machinery, 2001, pp. 120–127. ISBN: 1581133316. DOI: 10.1145/383952.383972. URL: https://doi.org/10.1145/383952.383972.

[44] J. Leonhardt. *Efficient and Explainable Neural Ranking | PhD thesis defense*. 2023. URL: https://mrjleo.github.io/slides/2023-phd/#/fast-forward (visited on 12/13/2023).

[45] Jurek Leonhardt et al. “Efficient Neural Ranking using Forward Indexes and Lightweight Encoders”. In: *ACM Trans. Inf. Syst.* (Nov. 2023). Just Accepted. ISSN: 1046-8188. DOI: 10.1145/3631939. URL: https://doi.org/10.1145/3631939.

[46] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. *Pretrained Transformers for Text Ranking: BERT and Beyond*. 2021. arXiv: 2010.06467 [cs.IR]. URL: https://arxiv.org/abs/2010.06467.

[47] Jimmy Lin et al. "Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 2356–2362. ISBN: 9781450380379. DOI: 10.1145/3404835.3463238. URL: https://doi.org/10.1145/3404835.3463238.

[48] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. "Distilling Dense Representations for Ranking using Tightly-Coupled Teachers". In: *CoRR* abs/2010.11386 (2020). arXiv: 2010.11386. URL: https://arxiv.org/abs/2010.11386.

[49] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. "In-Batch Negatives for Knowledge Distillation with Tightly-Coupled Teachers for Dense Retrieval". In: *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*. Ed. by Anna Rogers et al. Online: Association for Computational Linguistics, Aug. 2021, pp. 163–173. DOI: 10.18653/v1/2021.repl4nlp-1.17. URL: https://aclanthology.org/2021.repl4nlp-1.17/.

[50] Wenhao Lu, Jian Jiao, and Ruofei Zhang. "Twinbert: Distilling knowledge to twin-structured compressed bert models for large-scale retrieval". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 2645–2652.

[51] Yi Luan et al. "Sparse, Dense, and Attentional Representations for Text Retrieval". In: *Transactions of the Association for Computational Linguistics* 9 (2021). Ed. by Brian Roark and Ani Nenkova, pp. 329–345. DOI: 10.1162/tacl_a_00369. URL: https://aclanthology.org/2021.tacl-1.20/.

[52] Sean MacAvaney et al. "CEDR: Contextualized Embeddings for Document Ranking". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR'19. Paris, France: Association for Computing Machinery, 2019, pp. 1101–1104. ISBN: 9781450361729. DOI: 10.1145/3331184.3331317. URL: https://doi.org/10.1145/3331184.3331317.

[53] Sean MacAvaney et al. "Expansion via Prediction of Importance with Contextualization". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '20. Virtual Event, China: Association for Computing Machinery, 2020, pp. 1573–1576. ISBN: 9781450380164. DOI: 10.1145/3397271.3401262. URL: https://doi.org/10.1145/3397271.3401262.

[54] Sean MacAvaney et al. "Simplified Data Wrangling with ir_datasets". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 2429–2436. ISBN: 9781450380379. DOI: 10.1145/3404835.3463254. URL: https://doi.org/10.1145/3404835.3463254.

[55] Craig Macdonald and Nicola Tonellotto. "Declarative Experimentation inInformation Retrieval using PyTerrier". In: *Proceedings of ICTIR 2020*. 2020.

[56] Iain Mackie, Jeffrey Dalton, and Andrew Yates. "How Deep is your Learning: the DL-HARD Annotated Deep Learning Dataset". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 2335–2341. ISBN: 9781450380379. DOI: 10.1145/3404835.3463262. URL: https://doi.org/10.1145/3404835.3463262.

[57] Yury A. Malkov and Dmitry A. Yashunin. "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs". In: *CoRR* abs/1603.09320 (2016). arXiv: 1603.09320. URL: http://arxiv.org/abs/1603.09320.

[58] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521865719.

[59] Yusuke Matsui et al. "A survey of product quantization". In: *ITE Transactions on Media Technology and Applications* 6.1 (2018), pp. 2–10.

[60] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv preprint arXiv:1301.3781* (2013).

[61] Michael J. Neely. *Why we cannot divide by Zero*. URL: https://ee.usc.edu/stochastic-nets/docs/divide-by-zero.pdf.

[62] Tri Nguyen et al. "MS MARCO: A Human Generated MAchine Reading COmprehension Dataset". In: *CoRR* abs/1611.09268 (2016). URL: http://arxiv.org/abs/1611.09268.

[63] Rodrigo Nogueira and Kyunghyun Cho. "Passage Re-ranking with BERT". In: *CoRR* abs/1901.04085 (2019). arXiv: 1901.04085. URL: http://arxiv.org/abs/1901.04085.

[64] Rodrigo Nogueira et al. "Multi-stage document ranking with BERT". In: *arXiv preprint arXiv:1910.14424* (2019).

[65] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks". In: *CoRR* abs/1511.08458 (2015). arXiv: 1511.08458. URL: http://arxiv.org/abs/1511.08458.

[66] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[67] Marius-Constantin Popescu et al. "Multilayer perceptron and neural networks". In: *WSEAS Trans. Cir. and Sys.* 8.7 (July 2009), pp. 579–588. ISSN: 1109-2734.

[68] Martin F Porter. "An algorithm for suffix stripping". In: *Program* 14.3 (1980), pp. 130–137.

[69] Yingqi Qu et al. "RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Kristina Toutanova et al. Online: Association for Computational Linguistics, June 2021, pp. 5835–5847. DOI: 10.18653/v1/2021.naacl-main.466. URL: https://aclanthology.org/2021.naacl-main.466/.

[70] Nils Reimers and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks". In: *arXiv preprint arXiv:1908.10084* (2019).

[71] Stephen Robertson and Hugo Zaragoza. "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Foundations and Trends® in Information Retrieval* 3.4 (2009), pp. 333–389. ISSN: 1554-0669. DOI: 10.1561/1500000019. URL: http://dx.doi.org/10.1561/1500000019.

[72] Stephen E. Robertson et al. "Okapi at TREC-3". In: *Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994*. Ed. by Donna K. Harman. Vol. 500-225. NIST Special Publication. National Institute of Standards and Technology (NIST), 1994, pp. 109–126. URL: http://trec.nist.gov/pubs/trec3/papers/city.ps.gz.

[73] Koustav Rudra and Avishek Anand. "Distant Supervision in BERT-based Adhoc Document Retrieval". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. CIKM '20. Virtual Event, Ireland: Association for Computing Machinery, 2020, pp. 2197–2200. ISBN: 9781450368599. DOI: 10.1145/3340531.3412124. URL: https://doi.org/10.1145/3340531.3412124.

[74] Victor Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *CoRR* abs/1910.01108 (2019). arXiv: 1910.01108. URL: http://arxiv.org/abs/1910.01108.

[75] Harrisen Scells, Shengyao Zhuang, and Guido Zuccon. “Reduce, Reuse, Recycle: Green Information Retrieval Research”. In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’22. Madrid, Spain: Association for Computing Machinery, 2022, pp. 2825–2837. ISBN: 9781450387323. DOI: 10.1145/3477495.3531766. URL: https://doi.org/10.1145/3477495.3531766.

[76] Rayyan Shaikh. *Mastering BERT: A Comprehensive Guide from Beginner to Advanced in Natural Language Processing (NLP)*. Medium. 2023. URL: https://medium.com/@shaikhrayyan123/a-comprehensive-guide-to-understanding-bert-from-beginners-to-advanced-2379699e2b51.

[77] Jonathon Shlens. “Notes on Kullback-Leibler Divergence and Likelihood”. In: *CoRR* abs/1404.2000 (2014). arXiv: 1404.2000. URL: http://arxiv.org/abs/1404.2000.

[78] Karen Sparck Jones. “A statistical interpretation of term specificity and its application in retrieval”. In: *Document Retrieval Systems*. GBR: Taylor Graham Publishing, 1988, pp. 132–142. ISBN: 0947568212.

[79] Juan Terven et al. *Loss Functions and Metrics in Deep Learning*. 2024. arXiv: 2307.02694 [cs.LG]. URL: https://arxiv.org/abs/2307.02694.

[80] Nandan Thakur et al. “BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models”. In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021. URL: https://openreview.net/forum?id=wCu6T5xFjeJ.

[81] Ashish Vaswani et al. “Attention is all you need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.

[82] Qi Wang et al. “A comprehensive survey of loss functions in machine learning”. In: *Annals of Data Science* (2020), pp. 1–26.

[83] Shuai Wang, Shengyao Zhuang, and Guido Zuccon. “BERT-based Dense Retrievers Require Interpolation with BM25 for Effective Passage Retrieval”. In: *Proceedings of the 2021 ACM SIGIR International Conference on Theory of Information Retrieval*. ICTIR ’21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 317–324. ISBN: 9781450386111. DOI: 10.1145/3471158.3472233. URL: https://doi.org/10.1145/3471158.3472233.

[84] Shuai Wang and Guido Zuccon. “Balanced Topic Aware Sampling for Effective Dense Retriever: A Reproducibility Study”. In: *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’23. Taipei, Taiwan: Association for Computing Machinery, 2023, pp. 2542–2551. ISBN: 9781450394086. DOI: 10.1145/3539618.3591915. URL: https://doi.org/10.1145/3539618.3591915.

[85] Paul J Werbos. “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[86] Thomas Wolf et al. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[87] Yonghui Wu et al. “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation”. In: *CoRR* abs/1609.08144 (2016). arXiv: 1609.08144. URL: http://arxiv.org/abs/1609.08144.

[88] Lee Xiong et al. “Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=zeFrfgyZln.

[89] Shengyao Zhuang and Guido Zuccon. “Fast Passage Re-ranking with Contextualized Exact Term Matching and Efficient Passage Expansion”. In: *CoRR* abs/2108.08513 (2021). arXiv: 2108.08513. URL: https://arxiv.org/abs/2108.08513.

[90] Shengyao Zhuang and Guido Zuccon. “TILDE: Term independent likelihood moDEl for passage re-ranking”. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2021, pp. 1483–1492.

[91] Herbert Zuze and Melius Weideman. “Keyword stuffing and the big three search engines”. In: *Online Information Review* 37.2 (2013), pp. 268–286.

A

Overview of Architectures

See the following pages.

A.1. Fast-Forward

A.1.1. Original

See Figure A.1, which is described in chapter 2.

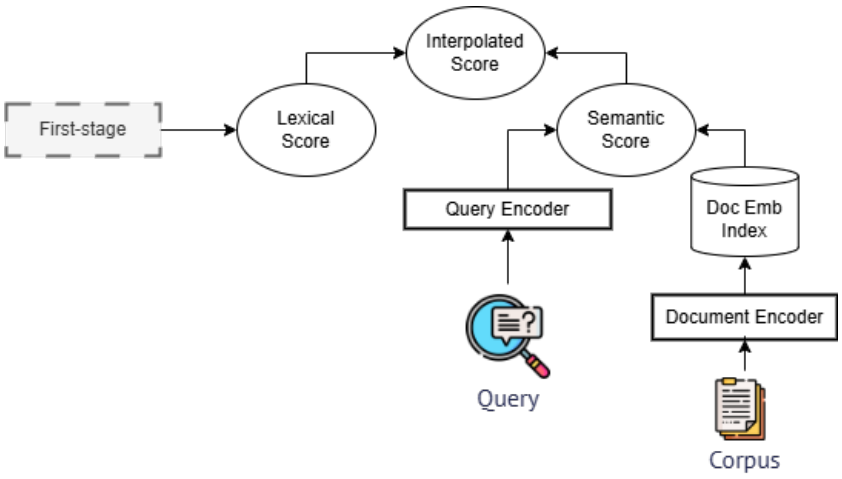


Figure A.1: Original architecture of neural re-ranking using Fast-Forward Indexes

A.1.2. Quantized

See Figure A.2. The document vectors are clustered and averaged into codewords, resulting in a significantly smaller index to fit in-memory at the cost of slightly lower performance.

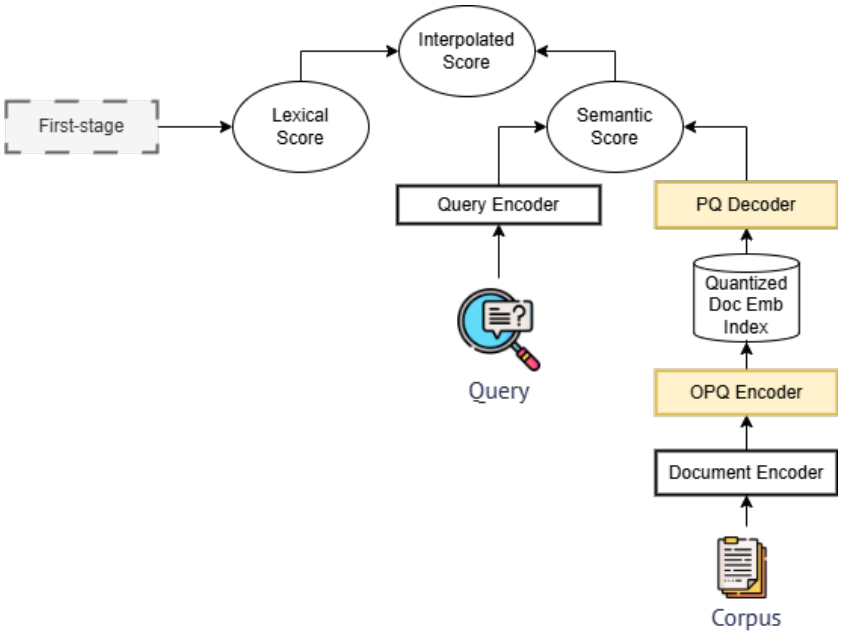


Figure A.2: Quantized architecture of neural re-ranking using Fast-Forward Indexes. Product quantization is highlighted in yellow.

A.2. Combined (AvgEmb + AvgTokEmb)

A.2.1. Initial

See Figure 3.5, which is described in subsection 3.1.2. This architecture combines the efficient encoders `AvgEmb` and `AvgTokEmb` into a multi-stage re-ranking architecture.

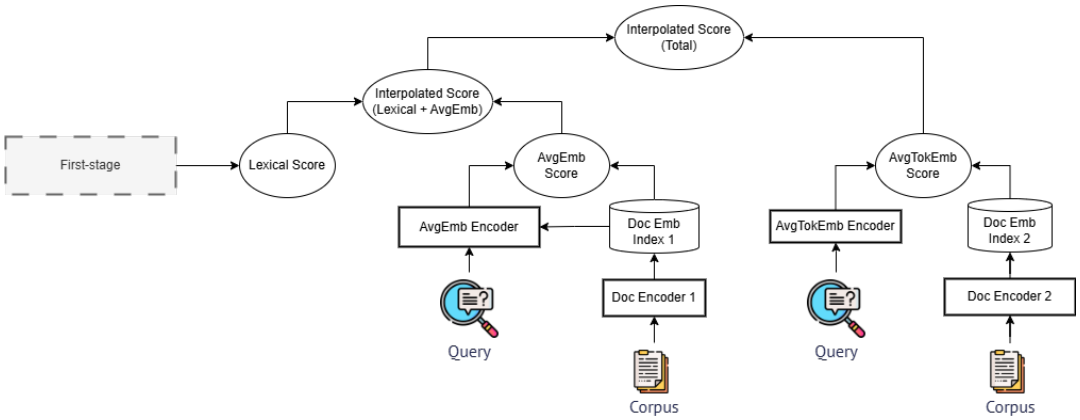


Figure A.3: Combined architecture of `AvgEmb` and `AvgTokEmb` encoders.

A.2.2. Refined

See Figure A.4, which is again described in detail in subsection 3.1.2. The combined architecture is refined to a simpler architecture with one shared document embedding index between the query encoders.

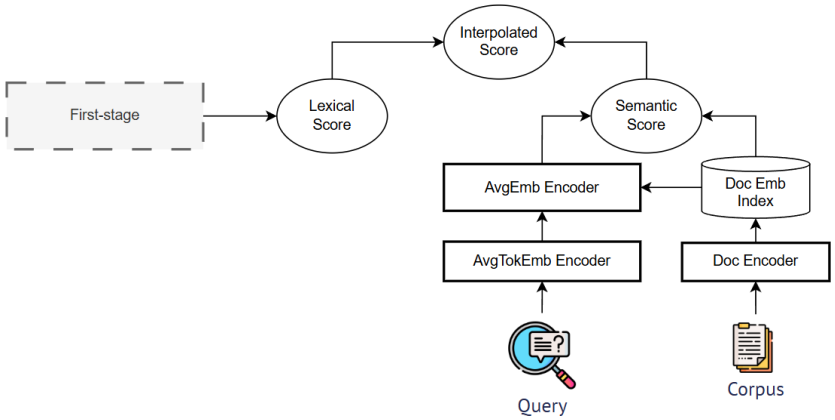


Figure A.4: Combined architecture of `AvgEmb` and `AvgTokEmb` encoders.

A.3. Training

See Figure A.5, described in section 3.2. Its limitations and alternative approaches are described in subsection 3.3.1.

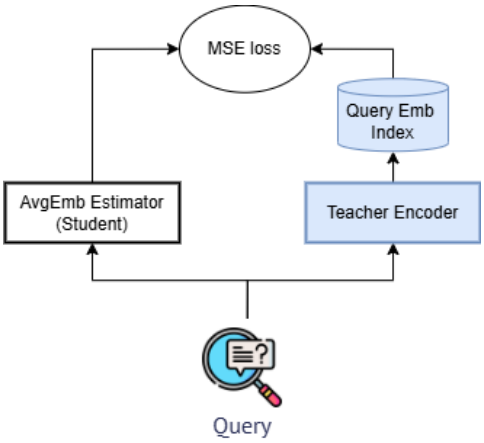


Figure A.5: Training architecture of `AvgEmb` estimator.

B

Examples with actual variables

B.1. AvgEmb Query Estimation

See Figure B.1 on the next page.

$$queries : \begin{bmatrix} q_1 : \text{"what is life"}, & q_2 : \text{"what now"} \end{bmatrix}$$

$$q\_tokens : \begin{cases} input\_ids : & \begin{bmatrix} q_1 : & t_a & t_b & t_c \\ q_2 : & t_a & t_d & t_{[PAD]} \end{bmatrix} & \in \mathbb{N}^{2\times 3\ (=batch\times max\_len)} \\ attention\_mask : & \begin{bmatrix} q_1 : & 1 & 1 & 1 \\ q_2 : & 1 & 1 & 0 \end{bmatrix} & \in \mathbb{N}^{2\times 3\ (=batch\times max\_len)} \end{cases}$$

$$q\_tok\_embs : \begin{bmatrix} q_1 : \begin{bmatrix} t_a : & -0.0013 & \cdots & 0.0145 \\ t_b : & -0.0358 & \cdots & -0.0293 \\ t_c : & -0.0255 & \cdots & 0.0245 \end{bmatrix} \\ q_2 : \begin{bmatrix} t_a : & -0.0013 & \cdots & 0.0145 \\ t_b : & 0.0241 & \cdots & -0.0397 \\ t_{[pad]} : & -0.0103 & \cdots & -0.0266 \end{bmatrix} \end{bmatrix} \in \mathbb{R}^{2\times 3\times 768\ (=batch\times max\_len\times dim)}$$

$$q\_tok\_weights : \begin{cases} \text{Init:} & \begin{bmatrix} q_1 : & 0.33 & 0.33 & 0.33 \\ q_2 : & 0.33 & 0.33 & 0.33 \end{bmatrix} \\ \text{Masked and normalized:} & \begin{bmatrix} q_1 : & 0.33 & 0.33 & 0.33 \\ q_2 : & 0.5 & 0.5 & 0 \end{bmatrix} \end{cases} \in \mathbb{R}^{2\times 3\ (=batch\times max\_len)}$$

$$q_{light} : \begin{bmatrix} q_1 : & (0.33 * -0.0013 + 0.33 * -0.0358 + 0.33 * -0.0255 = -0.0207) & \cdots \\ q_2 : & (0.5 * -0.0013 + 0.5 * 0.0241 = 0.0114) & \cdots \end{bmatrix}$$
$$= \begin{bmatrix} q_1 : & -0.0207 & \cdots & 0.0032 \\ q_2 : & 0.0014 & \cdots & 0.0126 \end{bmatrix} \in \mathbb{R}^{2\times 768\ (=batch\times dim)}$$

$$embs : \begin{bmatrix} q_1 : \begin{bmatrix} q_{light} & -0.0207 & \cdots & 0.0032 \\ d_1 : & -0.0013 & \cdots & 0.0145 \\ d_2 : & -0.0358 & \cdots & -0.0293 \\ d_3 : & -0.0255 & \cdots & 0.0245 \end{bmatrix} \\ q_2 : \begin{bmatrix} q_{light} & 0.0014 & \cdots & 0.0126 \\ d_1 : & -0.0013 & \cdots & 0.0145 \\ d_2 : & 0 & \cdots & 0 \\ d_3 : & 0 & \cdots & 0 \end{bmatrix} \end{bmatrix} \in \mathbb{R}^{2\times 4\times 768\ (=batch\times n\_embs\times dim)}$$

$$embs\_mask : \begin{bmatrix} q_1 : & 1 & 1 & 1 & 1 \\ q_2 : & 1 & 1 & 0 & 0 \end{bmatrix} \in \mathbb{N}^{2\times 4\ (=batch\times n\_embs)}$$

$$embs\_weights : \begin{cases} \text{Init:} & \begin{bmatrix} q_1 : & 0.7699 & 0.1039 & 0.0634 & 0.0628 \\ q_2 : & 0.7699 & 0.1039 & 0.0634 & 0.0628 \end{bmatrix} \\ \text{Masked and normalized:} & \begin{bmatrix} q_1 : & 0.7699 & 0.1039 & 0.0634 & 0.0628 \\ q_2 : & 0.8811 & 0.1189 & 0 & 0 \end{bmatrix} \end{cases} \in \mathbb{R}^{2\times 4\ (=batch\times n\_embs)}$$

$$\hat{q} : \begin{bmatrix} q_1 : & -0.0191 & \cdots & 0.0021 \\ q_2 : & 0.0011 & \cdots & -0.0109 \end{bmatrix} \in \mathbb{R}^{2\times 768\ (=batch\times dim)}$$

Figure B.1: Simplified example pass through the $\texttt{AvgEmb}_{\texttt{q,3-docs}}$ query encoder. A batch of 2 queries is used which have a maximum length of 3. Only 1 top-ranked document is retrieved for q_2, which is highly exceptional in practice.