Quantization for compact neural passage re-ranking

Catalin-Petru Lupau

Delft University of Technology

Quantization for compact neural passage re-ranking

by

Catalin-Petru Lupau

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on June 24, 2024 at 10:30 AM.

Thesis committee:

Chair:	Prof. Dr. Avishek Anand
Supervisor:	Dr. Jurek Leonhardt
Committee Member:	Prof. Dr. Kubilay Atasu
Project Duration:	November, 2023 - June, 2024
Faculty:	Electrical Engineering, Mathematics and Computer Science (EEMCS)
Student Number:	5042143

Preface

At the heart of every complex human civilization, there is a story. Or at least this is what the historian and writer Yuval Noah Harari claims. According to him, stories and myths make up the building blocks of complex human societies. At first glance, the concept might sound bizarre, alien even, but if we analyze things deeply it starts to make sense. This is because things such as corporations, money, governments, justice systems and even more abstract concepts like democracy and human rights are not 'real', they are not part of our physical reality. Would a government continue to exist if a majority of people would stop believing in its authority? As history has taught us times and times again, probably not. This is because institutions, laws, organizations and currency are part of an imagined reality, a shared set of ideas we all believe in. In other words, they are all pieces of information distributed across our brains.

In accordance with Harari's unorthodox claims, even the earliest complex civilizations have acknowledged the importance of centralizing, storing, retrieving and distributing information efficiently to people. Representative examples are the Great Library of Alexandria, established in the 3rd century BCE, the ancient Chinese imperial documentation going as far back as the Han Dynasty (202BC - 9AD) or the ancient Roman and Greek codices. What all of these large corpora of physical documents had in common was an index, i.e. a system that would allow a librarian to retrieve relevant pieces of information efficiently. But what happens when a civilization grows too complex and its library too big for an overworked ancient librarian to manage? If we were to trust Isaac Asimov and his novels, any human civilization that reaches a critical level of complexity will perish under its own weight, as its complexity becomes unmanageable.

Luckily, late 19th and 20th century visionaries such as Paul Otlet or Vannevar Bush had something different in store for us. They believed that informational complexity could be tamed by automation, so they proposed the creation of systems and machines such as the Mundaneum or the Memex, which are considered early visions of a contemporary search engine.

And here we are, at the beginning of the 21st century, when search engines have stopped being a novelty, but rather a mundane part of our everyday lives. Hype in our contemporary times is no longer generated by search engines but by AI systems powered by deep neural networks that can accomplish tasks that programmers of previous generations would have only dreamed about.

It is safe to say that, by now, artificial neural networks have made their way into almost every area of computer science. And search engines are no exception. If previous generations of search engines had to rely on lexical similarity to judge the relevancy of documents, now they can leverage the power of neural rankers to comprehend semantics. So far, neural ranking models seem like a dream come true that will lead us to the holy grail, the ultimate search engine that will empower our societies to scale their complexity indefinitely without ever collapsing. But, as Steve Jobs once said, "Sometimes life hits you in the head with a brick. Don't lose faith.". It turns out that dual-encoders, which represent the state-of-the-art neural ranking technology, require a lot of memory, which means that they don't scale well to large corpora of documents. Will we allow Jobs' bricks to shatter our dream of societies that scale indefinitely in complexity and knowledge? Or will we revive the spirit of Vannevar Bush and use technology to prevent the world from falling into a dark era of Asimovian proportions?

My thesis is a mathematical manifesto that directs the reader to march for the latter and help prevent the fall of humanity into the global Middle Ages. Since, as Suzanne Collins wrote, "Hope is the only thing stronger than fear". For some, it might sound like a utopia. My high school literature teacher once said that "any utopia put into practice is a dystopia". Well, the scope of my thesis is to prove her wrong. How, you might ask? The keyword is ***vector quantization****.*

Catalin-Petru Lupau
Delft, June 2024

Abstract

Passage re-ranking is a fundamental problem in information retrieval, which deals with reordering a small set of passages based on their relevancy to a query. It is a crucial component in various web information systems, such as search engines or question-answering systems. Modern approaches for building re-ranking systems rely on neural language models such as BERT [39], or its derivatives, to create dense indexes for the target document corpus. While such approaches bring significant performance gains compared to classical lexical re-rankers, they have the disadvantage of increased memory costs.

A family of methods that can be used to reduce the memory footprint of a dense index is called *vector quantization*. Vector quantization algorithms usually rely on a combination of clustering and space manipulation operations to perform a lossy compression of the dense index at the expense of index performance. While vector quantization is widely used for first-stage retrieval, its use in the context of re-ranking is underexplored. To this end, this thesis evaluates the effectiveness of product quantization, a well-known vector quantization method, on single-vector dual-encoders, specifically TCT-ColBERT [47] and Aggretriever [46]. In addition to this, we show how linear interpolation of sparse scores can be leveraged to improve the performance of quantized dense indices with negligible costs to the memory footprint or speed. Last but not least, we propose WolfPQ, a learnable quantization method aimed at further improving quantization for re-ranking by bridging the gap between the objective functions used in training product quantization and re-ranking systems, respectively.

Contents

List of Figures

List of Tables

Nomenclature

Quantizers

Abbreviation	Explanation
PQ	product quantization
OPQ	optimized product quantization
CQ	contextual quantization
JPQ	jointly optimized query encoder and product quantization
RepCONC	discrete representations via constrained clustering
DPQ	differentiable product quantization
WolfPQPre	pre-trained only WolfPQ
WolfPQ	WolfPQ with selection by minimum distance, fine-tuned using dense scores only
WolfPQSem	WolfPQ with selection by semantic sampling, fine-tuned using dense scores only
WolfPQI	WolfPQ with selection by minimum distance, fine-tuned using sparse-interpolated scores and mixed pointwise-listwise loss function
WolfPQSemI	WolfPQ with selection by semantic sampling, fine-tuned using sparse-interpolated scores and mixed pointwise-listwise loss function

Common notations

Symbol	Definition
ϕ	quantizer
$\phi^{(i)}$	subquantizer for subspace i
Φ	set of all theoretically possible quantizers
s	codebook index
C	codebook
$C^{(i)}$	subcodebook corresponding to the ith subspace
c	codeword
$c^{(i)}$	subcodeword corresponding to the ith subspace
x	vector
$x^{(i)}$	subvector corresponding to the ith subspace
$\hat{x}$	quantized vector
D	corpus
Q	query set
q	query
d	document (or passage, which is equivalent)
d^+	positive sample / relevant document for the query
d^-	negative sample / irrelevant document for the query
$\eta_d(d)$	semantic encoding of document d
$\eta_q(q)$	semantic encoding of query q

Symbol	Definition
$\hat{\eta}_d(d)$	quantized semantic encoding of document d
$\hat{\eta}_q(q)$	quantized semantic encoding of query q
H (or h)	the dimensionality of the vectors, encodings, embeddings, i.e. $\eta_d(d) \in \mathbb{R}^H$
M	the number of subspaces
K	number of codewords in a codebook
R	rotation matrix or rotation tensor
$R^{(i)}$	rotation matrix for subspace i (if R is a rotation tensor)

Note on abbreviation composition

A consistent abbreviation composition convention is used throughout the thesis to describe either one-stage or two-stage retrieval systems. The convention follows the structure described by equation 1.

$$\texttt{<Retrieval model>} + (\texttt{<Re-ranking model>}) - (\texttt{<Quantizer>}) - (\texttt{I}) \tag{1}$$

The pair of parentheses behind the components in the structure in equation 1 denotes that the component is optional. The optional $-I$ at the end encodes whether or not re-ranking by sparse interpolation is used. The list below provides a few examples of how the notation should be interpreted:

- **BM25 + Aggretriever-PQ-I**: BM25 retrieval combined with re-ranking performed by a PQ-quantized Aggretriever index that uses sparse interpolation.
- **TCT-ColBERT-PQ**: dense retrieval (with no re-ranking) performed by a PQ-quantized TCT-ColBERT index.
- **BM25 + ColBERT**: BM25 retrieval combined with re-ranking performed by ColBERT (no sparse interpolation used)
- **BM25 + TCT-ColBERT-WolfPQI**: BM25 retrieval combined with re-ranking performed by TCT-ColBERT quantized with a version of WolfPQ configured to perform selection by minimum distance. The version of WolfPQ was trained using interpolated dense scores and a mixed pointwise-listwise loss function. The re-ranking is performed without interpolation.
- **BM25**: plain BM25 retrieval
- **BM25 + Aggretriever-I**: BM25 retrieval combined with re-ranking performed by Aggretriever (not quantized) that uses sparse interpolation.

Note on core terms used interchangeably

The following table contains terms that are used interchangeably throughout the thesis.

Interchangeable terms	Explanation
centroid, codeword	*Codeword* is part of quantization vocabulary, while *centroid* is a term used in clustering. Since, in many quantization algorithms, the codewords are obtained by clustering, the two terms are used interchangeably.
set of centroids, codebook	Since a codebook is a set of codewords, the reason why the two are the same becomes self-explanatory.
document, passage	The difference between the two is subtle. Generally speaking, a document could be divided into multiple passages, and each passage could be encoded separately. Within the scope of the thesis, we assume all input is processed atomically. Hence, distinguishing between documents and passages does not make sense. So, the two terms are used as synonyms.
document set, passage set, corpus	All three terms refer to the set of documents we are searching through.

Interchangeable terms	Explanation
query, question	The term *query* is used in the context of search engines, while *question* is used in question-answering system. Since our research could be applied to both, the terms mean the same thing within the context of the thesis.
semantic encoding, semantic embedding, vector encoding, vector embedding, encoding, embedding, semantic vector	All of these terms refer to the vectors created by the dual-encoders to numerically represent the meaning of a document or a query.

1

Introduction

Deep learning has made its way into almost every area of computer science, and information retrieval is no exception. Previous generations of search engines had to rely on lexical similarity to determine the relevance of a passage or document with respect to some question or query [76, 71]. However, nowadays, we are able to accomplish the task with much better accuracy by relying on deep neural networks that can process the meaning (a.k.a semantics) behind a piece of text, be it passage or query [76, 71]. Even more, neural networks designed for natural language processing have made it possible to create chatbots that provide us directly with answers to our questions [52, 73]. Such chatbots are just glorified search engines [73], which make use of large language models to provide users with conversational interfaces, a representative example being Perplexity AI [2]. Regardless, it is safe to say that neural ranking models are becoming an increasingly important component of the technology stack powering search engines such as Google or Bing [38, 10].

We are probably all users of a popular web search engine by now. What a web search engine does is that it provides us with an *ordered* list of web pages *ranked* by their relevancy to our search query. Passage ranking is simply the task of *ordering a collection of passages in accordance to their relevancy to a query* [71]. Both the passages (documents) and the query are textual data. Hence, what we are doing is ordering longer snippets of human text (passages) based on how relevant they are with respect to the informational need contained in a shorter snippet of human text (query). A neural ranking model performs this task with the help of a neural network with inherent language modeling capabilities, known as a neural language model, that can understand and process the semantics behind the pieces of human text.

Dual-encoders, which are one of the main contemporary standards in neural ranking technology, require a lot of memory, which means that they do not scale well to large corpora of documents [44, 70, 72, 22]. To put things into perspective, according to *WorldWideWebSize.com*, a website using a transparent estimation methodology [7], the size of the indexed web at the moment of the writing is at least 5.1 billion pages [42]. Assuming that we make use of a single-vector dual-encoder such as TCT-ColBERT [47], and we encode every page using a single vector, which is unrealistic since many web pages would have to be split into multiple passages, the size of the index created by the model would be roughly 14591,2GB[1]. While an index of this can be stored given current technology, the index would live in secondary storage, which is slower than random access memory. And what about the future? As the number of web pages keeps increasing, so will the sizes of the neural indices we create, forcing us to expand the data centres. Nevertheless, data centres are expensive to maintain as they use much energy [45, 14], which we all know is not entirely produced in a sustainable manner [15].

Finding a solution to the problems highlighted above represents the primary motivation driving this thesis. Our aim is to reduce the memory footprint of dual-encoder-based dense indices, while maintaining as much of the original ranking performance. One category of techniques that comes to mind when

[1] $5.1 \cdot 10^9$(num. vectors) $\cdot$ 768 (vector dim.) $\cdot$ 4(32 bit values) $\cdot$ 1024^{-3}(B to GB conversion)

discussing the compression of vector collections is vector quantization, which has proven to be very effective in the field of approximate nearest neighbour search [53, 36].

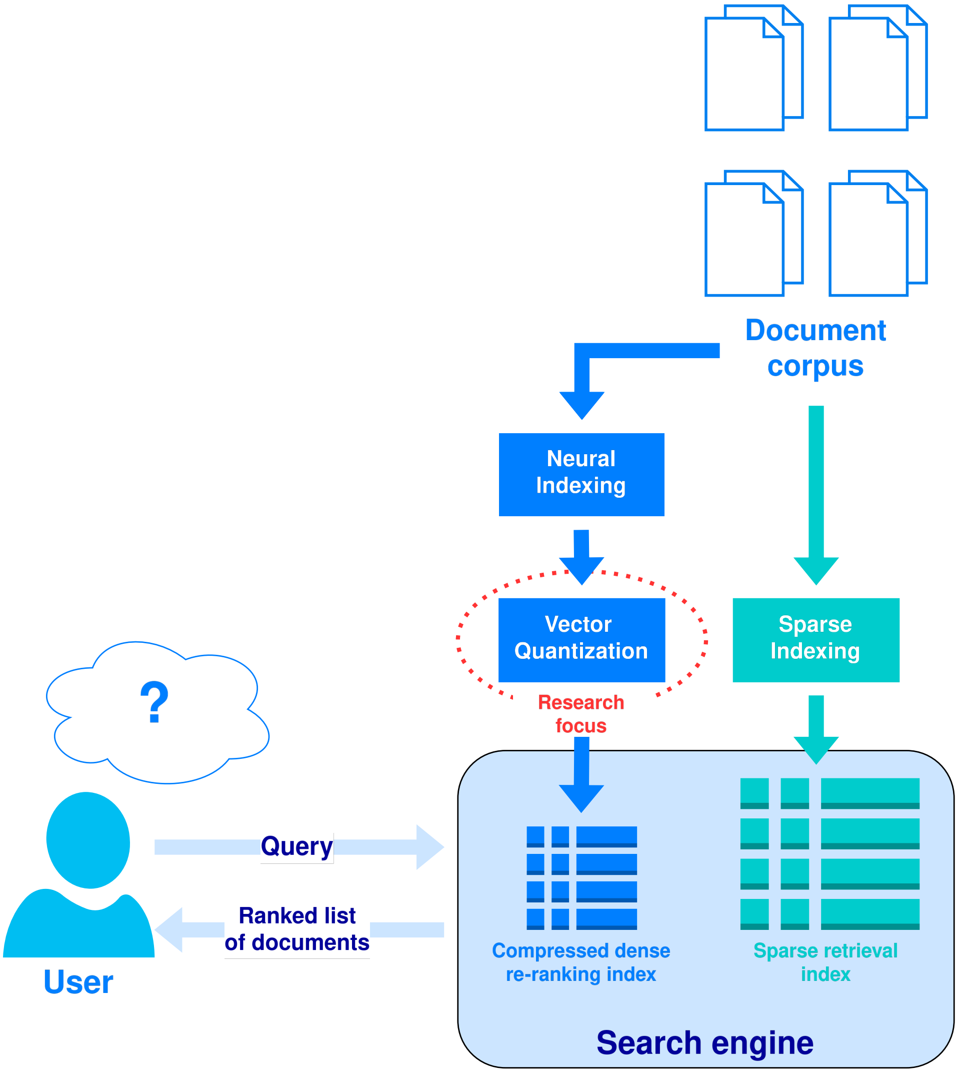


Figure 1.1: Broad overview of the problem setup and research focus.

Vector quantization is an umbrella term for a family of algorithms used to perform lossy compression of collections of vectors [53]. As the reader will hopefully discover in the next chapters of the thesis, dense indices, as created by dual-encoders, are nothing more than collections of semantic vectors that encode the meaning of the documents or passages in the corpus. This makes vector quantization algorithms a viable solution for compressing dense indices based on dual-encoders. While vector quantization has a long history as a technique being used to compress indices for retrieval (approximate nearest neighbour search), its use in the context of passage re-ranking is severely underexplored. To the best of our knowledge, there is only one paper introducing a quantization algorithm aimed at dual-encoder-based re-ranking. This algorithm is called a *contextual quantizer*, and it suffers from a major drawback [70].

The disadvantage of the contextual quantizer is that, by design, it is only applicable to dual encoders operating with term-level representations, such as ColBERT[70, 40]. As far as I am aware, there is no obvious way to adapt the contextual quantizer to single-vector dual encoders such as TCT-ColBERT [47] or Aggretriever [46], which create much smaller indices than ColBERT (or other term-level dual encoders) to begin with [47, 46, 40]. To this end, the thesis explores the use of vector quantization on single-vector dual-encoders for the purpose of improving the memory required by neural passage re-ranking. It does so by testing how well product quantization, one popular type of vector quantization, performs on TCT-ColBERT and Aggretriever. In addition to that, I explore the intersection between quantization and re-ranking with sparse interpolation, a technique recently proven to improve re-ranking performance [68, 44]. Finally, my thesis proposes WolfPQ, a novel quantization algorithm aimed to bridge the gap between the learning objectives of classical quantization and those of re-ranking models. All of the above contributions are made driven by the following research questions:

- **RQ1**: How well does product quantization work with single-vector dual encoders, specifically TCT-ColBERT and Aggretriever?
- **RQ2**: Can we increase the compression rate of product quantization, while maintaining the re-ranking performance, by introducing interpolation with sparse scores to the reranking score function?
- **RQ3**: Can we train a quantizer in a supervised fashion to optimize its behaviour for dense reranking? If so, can we train the quantizer in a way that enables it to adapt to sparse score interpolation achieving even higher performance improvements over product quantization?

The experimental results show that PQ-quantized single-vector dual encoders, specifically TCT-ColBERT and Aggretriever, achieve better *memory-performance trade-off* when compared against their token-level counterpart, BM25 + ColBERT-PQ. At the expense of slightly lower re-ranking performance, a quantized single-vector dual-encoder can use up to x10 less memory than its multi-vector counterpart.

In addition to the above, the sparse-retrieval-based configuration enables the use of sparse-score interpolation, which our experiments reveal to improve re-ranking performance by up to 224% at extremely high compression ratios.

Last but not least, WolfPQ is shown to generally outperform PQ at similar compression ratios. Comparisons to other ad-hoc retrieval methods indicate that our best setup (BM25 + TCT-ColBERT-WolfPQI-I) is a strong candidate for a state-of-the-art system when looking at performance and memory together. Further experimentation would be needed to claim that with high certainty.

In light of the above, the thesis is structured in 7 chapters, excluding this introduction. Chapter 2 provides the reader with the necessary background needed to comprehend the core of the thesis, while at the same time comprehensively summarizing the related work. Building on that, chapter 3 introduces the problem formally through the power of mathematical modelling by rigorously defining all concepts that make up the theoretical framework of the research. Chapter 4, on the other hand, contains detailed summaries of three scientific papers that were foundational to this research, particularly due to being sources of inspiration for the design of WolfPQ, the novel quantization algorithm that this thesis proposes. Chapter 5 makes a deep dive into the methodology by explaining the techniques used to answer the research questions and, more importantly, explaining the inner workings of WolfPQ. Chapter 6 provides details about the experimental setup, which are crucial for the reproducibility of the results. Finally, chapters 7 and 8 present the results and the conclusion aimed to direct future scientific endeavours.

2

Background and related work

The purpose of this chapter is to provide the reader with the theoretical background needed to fully comprehend the reasons behind the proposed research, its scope, as well as its contributions within the broader fields of information retrieval and natural language processing. To this end, the chapter is logically structured into three main sections that correspond to the core concepts needed for a complete understanding of the thesis.

The first section explores the retrieval pipeline, a broad architectural pattern commonly used by search engines whose understanding is important for correctly framing the topic of the thesis within the wider field of information retrieval. Consequently, the second section takes a deep dive into the problem of passage ranking, one of the main components of the retrieval pipeline, by providing an overview of the evolution of relevant techniques and their relationship to breakthroughs in deep learning. Finally, the third section explains the concept of vector quantization, a well-known compression technique which, in the context of this research, is used with the goal of improving passage ranking.

2.1. The two-stage retrieval pipeline

Ad-hoc passage ranking is a core component of a modern search engine. The task consists of retrieving and ranking passages from a corpus based on their relevancy to a query specified by the user [71]. Most solutions for ad-hoc document ranking rely on a common architectural framework known as the *two-stage retrieval pipeline*. As indicated by its name, the two-stage retrieval pipeline involves multiple steps and components. Since the retrieval system has to search over a large corpus of passages (or documents), the need for a data structure that enables efficient search over a corpus arises. Such a data structure is known as an *index*, and the process of constructing an index is called *indexing*. It is thus not a coincidence that the very first step of the retrieval pipeline is indexing. The index is always pre-computed and used from memory during runtime [71].

The process through which the retrieval system returns the relevant passages to the user is a two-stage process. Initially, the query is sent to the index, which returns a set of documents that are likely to be related to the query. This set is formally known as the *retrieval set* or *first-stage retrieval set*. As corpora can be huge, it is important that *first-stage retrieval* is very fast, even at the expense of being precise. Subsequently, the small set of retrieved passages is ordered based on query relevancy. This enables the system to discard the least relevant passages and present the most relevant passages in a convenient order to the user. The process of re-ordering the passages in the first-stage retrieval set is called *re-ranking*. Unlike *first-stage retrieval*, *re-ranking* needs to be very accurate despite being slower.

What is common across both first-stage retrieval and re-ranking is the need to have a relative measure of query relevancy for passages, be it to decide which documents to include in the retrieval set or to re-order the documents based on how related they are to the query. The process of assigning a score based on which passages can be compared in terms of their relevancy to a query is known as *passage ranking* or *text ranking* [71]. Information retrieval literature contains numerous techniques for passage ranking, each with its own benefits and drawbacks. A summary of these techniques will be provided in the following sections.

2.2. Passage ranking

The previous section has provided an overview of a fundamental architectural pattern in information retrieval, the two-stage-retrieval pipeline, while simultaneously introducing passage ranking, a core component of the pipeline. Following the chain of reasoning, this section will go deeper into passage ranking by looking at the three most popular families of techniques and their relative advantages and drawbacks.

As previously mentioned, the purpose of passage ranking is to assign numerical scores to the passages in a collection based on which their relative relevancy to the query can be compared. Thus, the following subsections will present various methods that can be used to assign such scores.

2.2.1. Sparse rankers

One of the older categories of passage ranking techniques still used nowadays is constituted by sparse rankers [71]. Before the deep-learning era, which enabled us to create semantic models of human language and text, sparse rankers used to be the state-of-the-art, having powered many of our legacy search engines [71, 64].

While differing from each other in terms of mathematical specifics, sparse rankers share a plethora of common characteristics, which justifies their classification as a distinct methodological category. Among the commonalities of sparse rankers, we can enumerate the reliance on classical inverted indexes that store term-level statistics about the corpus, analyzing text lexically (rather than semantically), leading to an inability to properly deal with synonyms (vocabulary mismatch [25], [75]), as well as a theoretical foundation built upon probability theory applied on sets of terms (words).

Maybe one of the simplest forms of sparse rankers is the *TD-IDF* method [3]. The intuition behind *TD-IDF* is rewarding passages in which the query terms appear with a high frequency while assigning a lower weight to frequent terms that are common across the entire corpus. Formulating this idea mathematically leads to equation 2.1. The formula in equation 2.1 can also be obtained by treating the problem from an information-theoretical perspective, in which case the TD-IDF weights can be interpreted as the expected mutual information between a passage in the corpus and some word in some arbitrary query [3].

$$\begin{aligned} & w_{i,j} := tf_{i,j} \cdot \log(\tfrac{|D|}{df_i}) \\ & w_{i,j} - \text{tf-idf weight between query word i and passage j} \\ & tf_{i,j} - \text{frequency of word i in passage j} \\ & df_i - \text{number of passages in the corpus that contain word i} \\ & |D| - \text{the size of the corpus} \end{aligned} \tag{2.1}$$

The same intuition as TF-IDF is used by most sparse rankers. *BM25* (equation 2.2), one of the most popular sparse ranking functions, can be seen as a refinement of TF-IDF introducing term saturation, which means that the contribution of a term to the total relevancy of a document decreases with each new occurrence of the term [61]. Despite being an older method, BM25 is still used nowadays, and it is a very common benchmark in information retrieval research [44, 47, 46, 40]. We will go deeper into BM25 in the next chapter, as it represents an integral part of the problem setup.

$$\begin{array}{l}
bm25(d,q) = \sum_{w \in q} \frac{f_{w,d} \cdot (k_1+1)}{f_{w,d}+k_1 \cdot (1-b+b \cdot \frac{|d|}{\texttt{avgdl}})} \cdot \log \frac{|D|-n_w+0.5}{n_w+0.5} \\
f_{w,d} := \text{the frequency of word } w \text{ in document } d \\
n_w := \text{number of documents in the corpus containing word } w \\
|D| := \text{the size of the corpus} \\
|d| := \text{the length of document } d \\
\texttt{avgdl} := \text{the average length of a document in the corpus} \\
b := \text{paramater controlling the impact of document length on word (term) saturation} \\
k_1 := \text{parameter controlling word (term) staturation}
\end{array} \tag{2.2}$$

As previously mentioned, since sparse rankers such as TD-IDF and BM25 analyze passages and queries only lexically, they perform poorly in cases where the passage and a query do not use the exact words to describe the concept but rather synonyms. This problem, called the vocabulary mismatch problem [25, 75], is what motivated the development of more advanced sparse rankers such as SPLADE [24] or DeepImpact [51], as well as dense rankers [71] (*cross-encoders* and *dual-encoders*), which will be presented in the following subsections.

2.2.2. Cross-encoders

The recent developments in natural language processing, powered by deep learning, have led to the development of neural language models [54]. The idea behind neural language models is to train neural networks on large amounts of human text to enable them to understand and generate human language [54]. This data-driven approach towards language modelling represented a step forward from the sparse rankers of the previous subsection since they provided a way to mitigate the vocabulary mismatch problem, which used to be the main obstacle in the way of developing more accurate passage rankers.

The first neural network architectures that were used for the purpose of processing language were RNNs, LSTMs and GRUs [58, 49, 62]. These families of neural network architectures enjoyed relative success but suffered from two main limitations. Firstly, recurrent neural networks have the inability to deal with very long sequences of text, a problem formally known as the *vanishing gradient problem* [65, 43, 55]. Secondly, all of the mentioned neural network architectures are sequential, which means that training them on a lot of data parallelly is not possible [66].

Everything changed with the introduction of the transformer [66], a foundational model that enables efficient training of neural language models on huge amounts of data, tackling the limitations of the recurrent neural networks. Using the transformer, highly effective neural language models, known as large language models, were developed. One of the most popular such large language model among natural language processing researchers is BERT [39], which has the property of analyzing text in a bi-directional fashion.

Taking into account all of the above, the question that arises is whether or not large language model technology can be leveraged to improve passage ranking. Cross-encoders are a family of passage rankers that aim to achieve exactly that, outperforming sparse rankers by a significant margin [71]. Cross-encoders work by feeding a query-passage pair into a large language model with the purpose of obtaining a relevance score. To illustrate this, we will have a look at a widely popular BERT-based cross-encoder known as MonoBERT [56].

The main idea behind MonoBERT [56] is to convert the passage ranking problem into a binary text classification problem, where the two labels are *relevant* and *non-relevant*. The passages are then ordered according to the relevancy probabilities predicted by the model, in accordance with the *probability ranking principle*. As illustrated by figure 2.1, the architecture of MonoBERT consists of an instance of the BERT large language model, followed by a shallow neural network used for binary classification. The query q and the passage d are truncated to 64 and 512 tokens, respectively, and then concatenated. The concatenated input is propagated through the BERT model and the [CLS] vector of the output is extracted. The [CLS] vector is then further propagated through the shallow feed-forward neural network, which finally outputs the ranking probability. The inference process is formally shown in equation 2.3.

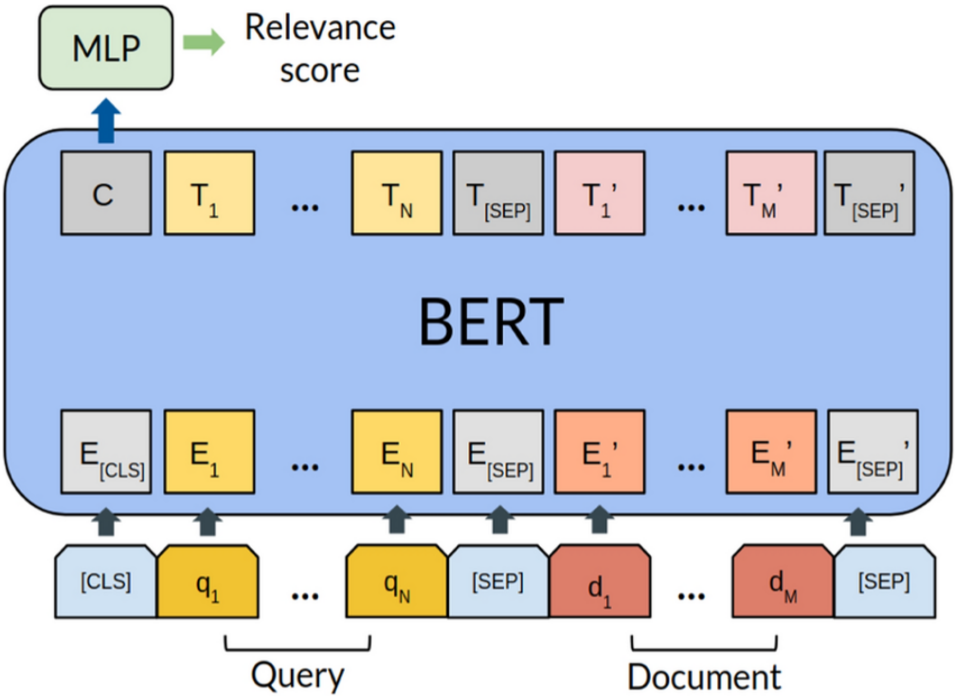


Figure 2.1: Diagram illustrating the architecture of monoBERT. Figure taken from [71].

$$
\begin{aligned}
&input := \texttt{trunacate}(q, 64) \circ \texttt{trunacate}(d, 512)\\
&monoBERT(d, q) = P(d|q) := FeedForwardNet(BERT(input)_{[CLS]})\\
&\\
&\texttt{trunacate}(x, n) - \text{truncates the textual input } x \text{ to } n \text{ tokens}\\
&a \circ b - \text{concatenation between vectors } a \text{ and } b\\
&BERT(x)_{[CLS]} - \text{the [CLS] vector obtained by feeding x into BERT}
\end{aligned}
\tag{2.3}
$$

The instance of the BERT model used in MonoBERT is initialized with the weights corresponding to one of the instances of pre-trained BERT models. MonoBERT is then further trained in an end-to-end fashion (BERT and the feed-forward neural network together) on query-passage pairs obtained by performing passage selection from top BM25 results. During the training process, the cross-entropy loss is used, as shown in equation 2.4, which is a standard loss function for binary classification problems.

$$
\begin{aligned}
&L_{monoBERT}(q) = -\textstyle\sum_{d^+ \in S^+(q)} \log P(d^+|q) - \textstyle\sum_{d^- \in S^-(q)} (1 - \log P(d^-|q))\\
&\\
&S^+(q) - \text{positive samples for query q}\\
&S^-(q) - \text{negative samples for query q}
\end{aligned}
\tag{2.4}
$$

As we have seen in the section about the two-stage retrieval pipeline, retrieval speed is of high importance for any information retrieval system [32, 37]. Considering this, a major drawback of monoBERT in particular and cross-encoders in general becomes apparent. Having to process all passages at runtime for every new incoming query is very time-consuming [71]. This results from the fact that, as already mentioned, passage corpora can be quite large, as well as from the fact that passages can contain many tokens. Since transformers, the building blocks of most cross-encoders, have a quadratic time complexity with respect to the number of input tokens [66], the processing time of applying cross-encoders for passage re-ranking can easily skyrocket. Hence, cross-encoders are not the most feasible technique for real-time information retrieval [71].

2.2.3. Dual-encoders

Dual-encoders were born out of the necessity to make cross-encoders more feasible for real-time passage ranking by reducing their time complexity while at the same time preserving as much of their superior ranking performance [37, 71]. This was achieved by tackling the most obvious bottleneck of the cross-encoder ranking architecture, i.e. the necessity to process passages for every new incoming query. Hence, the idea behind dual-encoders is to split the processing of queries and passages into two distinct operations. In this way, the passages can be pre-computed before runtime and retrieved from memory when needed. Thus, dual-encoders follow the classical algorithmic design technique of trading memory for increased processing speed. To better illustrate the workings of dual-encoders, we will have a look at ColBERT [40], a well-known dual-encoder among IR researchers, as well as TctColBERT [47] and Aggretriever [46], which represent the focus of our research.

ColBERT

ColBERT [40] is a widely popular dual-encoder that, similarly to monoBERT [56], is built upon the BERT [39] neural language model. Following the dual-encoder architectural framework, ColBERT splits the computation of the passage and the query into two by introducing a query encoder and a passage/document encoder. Both the query encoder and the document encoder share the same instance of BERT, but the two input categories are differentiated by prepending a special token [Q] to queries and [D] to passages. The query and document encoders have very similar architectures, as both preprocess the input text using BERT and then further propagate the output through a convolutional neural network followed by a normalization step. The output of the document encoder pipeline is further put through a filtering step that removes the embeddings corresponding to punctuation marks, which are considered semantically irrelevant. The workings of the two encoders are described in more mathematical depth by equation 2.5.

$$
\begin{aligned}
&E_q := Normalize(CNN(BERT(\texttt{truncate}([Q] \circ q, N_q))))\\
&E_d := Filter(Normalize(CNN(BERT([D] \circ d))))\\
&\\
&E_q - \text{bag of ColBERT embeddings for the query}\\
&E_d - \text{bag of ColBERT embeddings for the document}\\
&N_q - \text{number of tokens to truncate the query to}\\
&\texttt{trunacate}(x, n) - \text{truncates the textual input } x \text{ to } n \text{ tokens}\\
&a \circ b - \text{concatenation between vectors } a \text{ and } b
\end{aligned}
\tag{2.5}
$$

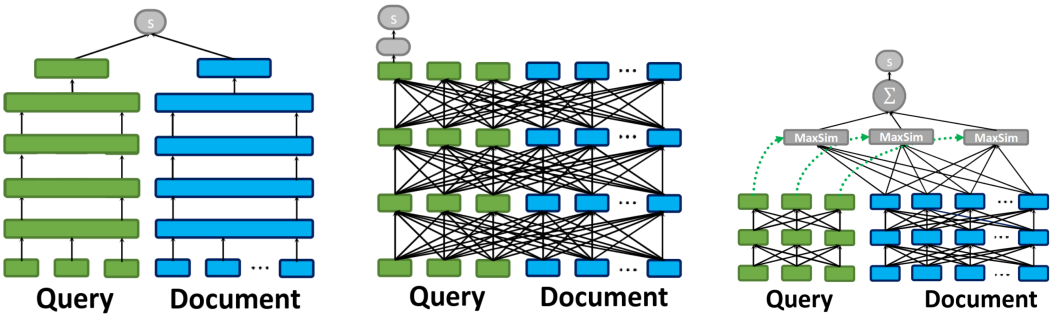


Figure 2.2: Comparison between the *representation learning* paradigm (left), the *all-to-all interaction* paradigm used by cross-encoders (middle) and the *late interaction* paradigm (right). Figures taken from [40].

Maybe the most interesting part about ColBERT is the introduction of the concept of *late-interaction* [40]. A dual-encoder that works purely within the realm of *representation learning* would summarize both the query and the passage with vectors that belong to the same semantic space and would compute the ranking score based on some similarity metric between the vectors. This would lead to a significant decrease in ranking performance since the passage and the query are never analyzed together by the neural network, or, in other words, the two never interact at the level of neural connections. At the other end of the spectrum, we have the cross-encoders that analyze the query and the document together from the very beginning, achieving high ranking performance but low processing speeds. Models like

ColBERT, which operate in the paradigm of *late-interaction*, try to achieve a middle ground by separating most of the query and passage processing but still introducing some interaction between the two inside the final scoring function. For ColBERT, this is achieved through an operator called *MaxSim*, that computes the relevancy score based on the two bags of embeddings outputted by the query encoder and document encoder, respectively, as shown in equation 2.6 and figure 2.3.

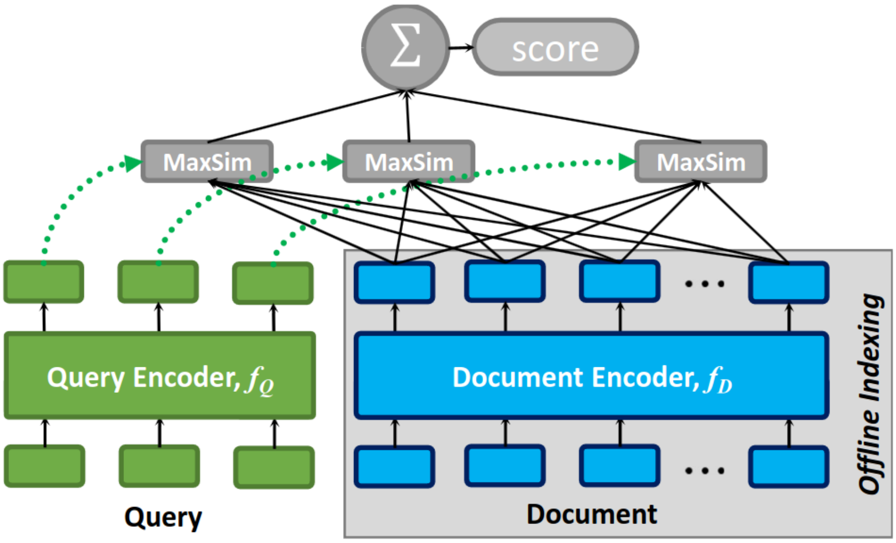


Figure 2.3: Diagram illustrating the architecture of ColBERT. Figure taken from [40]

$$colBERT(d,q) := maxSim(E_d, E_q) = \textstyle\sum_{e_q \in E_q} \max_{e_d \in E_d} e_q \cdot e_d^T$$

$$\begin{aligned} &e_q - \text{query embedding} \\ &e_d - \text{document embedding} \\ &E_q - \text{bag of query embeddings outputted by the query encoder} \\ &E_d - \text{bag of document embeddings outputted by the document encoder} \end{aligned} \tag{2.6}$$

TCT-ColBERT

While achieving state-of-the-art results [40], ColBERT has the disadvantage of requiring a lot of memory. This is because, as illustrated by equation 2.5, ColBERT outputs and stores a bag of embeddings for each passage in the corpus. TCT-ColBERT [47] and Aggretriever [46] represent attempts to mitigate this by creating single-vector representations for the passages. In essence, the two models are attempts to switch back from the *late-interaction* paradigm to the *representation learning* paradigm while trying to preserve as much of the ranking performance specific to late-interaction.

In the case of TCT-ColBERT the transition back from *late-interaction* to *representation learning* is achieved by means of knowledge distillation. More specifically, TCT-ColBERT is a distilled version of ColBERT that replaces the *maxSim* operator by the dot product between the single-vector representations of the query and the passage. To this end, TCT-ColBERT defines two probability distributions expressing the relevance of documents given queries, a teacher distribution estimated using ColBERT and a student distribution estimated using TCT-ColBERT, as shown in equations 2.7 and 2.8. TCT-ColBERT is, hence, trained in such a way that the student distribution approaches the teacher distribution, which corresponds to minimizing the Kullback–Leibler divergence [41] between the two probabilities, as expressed by equation 2.9.

$$\hat{P}(d|q) = \frac{\exp(maxSim(E_d,E_q)/\tau)}{\sum_{d' \in D}\exp(maxSim(E_{d'},E_q)/\tau)} \tag{2.7}$$

$\hat{P}(d|q)$ – relevancy probability of passage d estimated using ColBERT
τ – temperature parameter

$$P(d|q) = \frac{\exp(Pool(E_d) \cdot Pool(E_q))}{\sum_{d' \in D}\exp(Pool(E_{d'}) \cdot Pool(E_q))}$$
$$Pool(E) := \frac{1}{|E|}\sum_{e \in E} e \text{ or } \max_{e \in E} e \tag{2.8}$$

$P(d|q)$ – relevancy probability of passage d estimated using Tct-ColBERT

$$KL(\hat{P}||P) = \sum_{d' \in D} \hat{P}(d'|q) \log \frac{\hat{P}(d'|q)}{P(d'|q)} \tag{2.9}$$

Aggretriever

Unlike TCT-ColBERT, which achieves effective single-vector representations by performing knowledge distillation from ColBERT, Aggretriever [46] is a single-vector dual-encoder built from scratch with BERT as a basis. Aggretriever is constructed on the intuition that simply relying on the [CLS] token of BERT as a single-vector representation for a passage is not enough, as the token contains insufficient information. Instead, it proposes the aggregation of the MLM output of BERT into a vector named $agg*$, which, once combined with the [CLS] token, forms the single-vector representation of the textual input.

The $agg*$ vector is obtained by applying a series of operations on the tokens outputted by the MLM head of BERT. First, the MLM head of BERT is re-used in combination with softmax to project each individual token into a high-dimensional vector representing a probability distribution over the vocabulary of BERT, as indicated by equation 2.10.

$$p_{q_i} := \textit{softmax}(e_{q_i} \cdot W_{MLM} + b_{MLM}) \tag{2.10}$$

e_{q_i} – the ith BERT output token
W_{MLM}, b_{MLM} – the weights and bias of the pre-trained MLM linear projector
p_{q_i} – probability distribution corresponding to token q_i

The following step is the application of weighted max-pooling on the sequence of high-dimensional vector representations corresponding to each of the tokens. The weights used in max-pooling are obtained by propagating the output tokens once more through a one-layer neural network. The procedure is described in equation 2.11.

$$v_q[j] = \max_{i \in \{1,2,\ldots,l\} w_i \cdot p_{q_i}[j]}$$
$$w_i = |e_{q_i} \cdot W + b| \tag{2.11}$$

$v_q[j]$ – result of the pooling operation at the jth position inside the vector
W, b – the weights and bias of the single vector neural network
$p_{q_i}[j]$ – the value of the probability vector corresponding to token q_i at index j

The $agg*$ vector is obtained from the output of max-pooling v_q by applying a complex pooling operation that consists of creating multiple subdivisions of the input and taking the maximum in each subdivision. Finally, the single-vector representation for both input queries and input passages is formed by concatenating the $agg*$ vector with the $[CLS]$ token outputted by BERT. A conceptual diagram of how Aggretriever works is illustrated by figure 2.4.

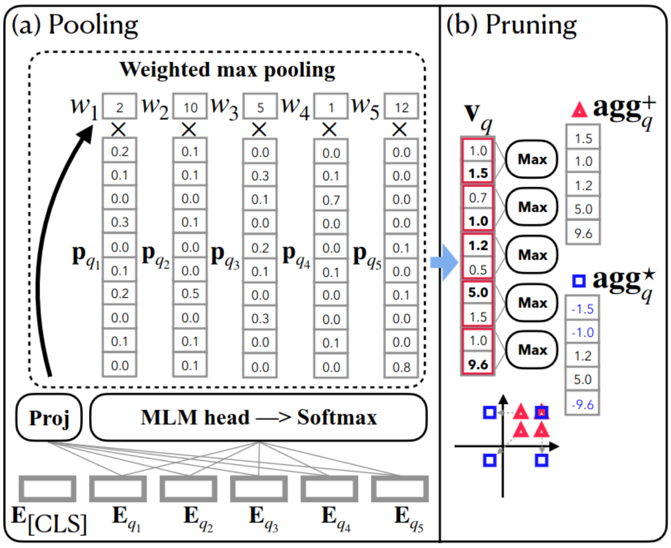


Figure 2.4: Diagram illustrating the architecture of Aggretriever. Figure taken from [46].

2.3. Vector quantization

In the previous sections, we have provided an overview of the two-stage retrieval pipeline as well as a summary of the evolution of the algorithms used in one of its key components, namely passage ranking. We have also seen that single-vector dual-encoders, such as TCT-ColBERT [47] and Aggretriever [46], provide a good balance between memory, speed and ranking performance, making them strong candidates for the re-ranking component of the retrieval pipeline [44]. As mentioned in the introduction, the focus of the thesis is the application of vector quantization, a family of lossy compression algorithms, on the dense indices corresponding to single-vector dual-encoders with the purpose of further reducing memory usage. To this end, this section will present the reader with a general description of the technique, which will form a necessary basis for comprehending the methodological section of this thesis.

2.3.1. General formulation

Vector quantization represents a family of algorithms that aim to perform lossy compression of vector spaces [53]. These algorithms have been studied extensively in the context of achieving efficient approximate nearest neighbour search (ANN). However their use for the purpose of passage ranking is severely underexplored [70].

From a mathematical perspective, the classical forms of vector quantization find a mapping between a *continuous* vector space and a *discrete* vector space. The mappings that these algorithms find need to be in such a way that the distance (a.k.a. distortion) between the original vector and its representation in the discrete space is minimized [53, 36]. Concretely, given a sample $X = \{x_1, x_2, ..., x_n\} \subset \mathbb{R}^H$ of n H-dimensional real vectors, the algorithm yields a new finite set of vectors $C = \{c_1, c_2, ..., c_k\} \subset \mathbb{R}^H$ $(k < n)$ representing the discrete vector space and a mapping $\phi : \mathbb{R}^H \to C$, encoding any real vector from the continuous space as one of the vectors $c_i \in C$ in the discrete space. The set C is formally known as a **codebook**, and each individual vector $c_i \in C$ is called a **codeword**. The mapping $\phi : \mathbb{R}^H \to C$, associating real-valued vectors to codewords, is called a **quantizer**.

The quality of a quantizer is measured using a function known as **distortion** [53, 36]. Since the distortion needs to express how different the quantized version of a vector is from its original form, it is mathematically defined by the euclidean distance between the quantized and the non-quantized versions of the vector, as expressed in equation 2.12.

$$\begin{aligned}&\texttt{dist}(x, \phi(x)) = \|x - \phi(x)\|^2\\&\texttt{dist}(x, \phi(x)) - \text{the distortion between the original vector } x \text{ and its quantized form } \phi(x)\end{aligned} \tag{2.12}$$

When the distortion is low, the discrete vector space represented by the codebook C is a good approximation of the continuous vector space from which we have sampled X. Thus, the objective of an (unsupervised) quantization algorithm is to find a codebook C and a quantizer $\phi : \mathbb{R}^H \rightarrow C$ such that the mean distortion is minimized, as shown in equation 2.13.

$$\begin{aligned}&MD(X; \phi^*, C^*) = \tfrac{1}{m} \textstyle\sum_{x_i \in X} \texttt{dist}(x_i, \phi^*(x_i))\\&\phi, C = \text{argmin}_{\phi^* \in \Phi, C^* \subset \mathbb{R}^H} MD(X; \phi^*, C)\\[1em]&MD(X; \phi^*, C^*) - \text{mean distortion induced by the quantizer } \phi^* \text{ on the vector dataset X}\\&\Phi - \text{the set of all possible quantizers}\\&\phi - \text{the optimal quantizer}\\&C - \text{the optimal codebook}\end{aligned} \tag{2.13}$$

One interesting aspect to notice is that despite the fact that our quantizer $\phi : \mathbb{R}^H \rightarrow C$ is defined on the entire space of H-dimensional real vectors $\mathbb{R}^H$, it is only obtained using a finite subset of vectors $X \subset \mathbb{R}^H$. The reason for this is that the subset X is assumed to be a *representative random sample* of the vector space, which can be conceptually thought of as a multi-dimensional probability distribution despite not being concretely modelled mathematically. In other words, under its classical formulation, vector quantization can be seen as an *unsupervised learning algorithm*. From the perspective of this analogy, the subset X is the training set of the learning algorithm, while the mean distortion MD in equation 2.13 is the loss function. Last but not least, the quantizer ϕ and the codebook C make up the model that we are learning, which we hope to generalize well for new vectors outside the training dataset X. The analogy between vector quantization and unsupervised learning is summarized in table 2.1.

Vector Quantization	**Unsupervised Learning**
$\phi : \mathbb{R}^H \rightarrow C$ (quantizer)	model
X (sample from continuous vector space)	training set
$MD(X; \phi, C)$ (mean distortion)	loss function (MSE loss)

Table 2.1: Correspondance between concepts in vector quantization and unsupervised learning.

2.3.2. Unsupervised quantization

Having seen the general formulation of vector quantization in the previous subsection, we will now move on to presenting a few concrete, commonly used vector quantization algorithms whose understanding is vital to develop the intuition needed to comprehend the methodological part of the thesis. The three quantization algorithms that we will present build on top of each other, which helps illustrate how quantization can be developed and iteratively improved by leveraging concepts such as *clustering*, *subspace division* and *space transformations*.

K-means quantization

One of the most basic forms of vector quantization, which is essentially just a subcase of product quantization (PQ) [36, 53], is K-means quantization. The idea behind K-means quantization is elementary yet very powerful. K-means clustering [1] is applied on the input vector dataset X. This leads to the emergence of K vector clusters, each cluster i having its own centroid c_i. We will compose the codebook C out of the centroids of all clusters formed by applying K-means, as shown in equation 2.14.

$$\begin{aligned}
&\{c_1, c_2, ..., c_K\} \leftarrow \texttt{KMeans}(X) \\
&C := \{c_1, c_2, ..., c_K\} \\
&\\
&C - \text{codebook} \\
&c_i \in \mathbb{R}^H - \text{codeword / centroid} \\
&X - \text{input vector dataset (training set)}
\end{aligned} \tag{2.14}$$

Having fixed our codebook, the definition of the quantizer function follows through from the fact that our objective is to minimize the mean distortion, as expressed by equation 2.13. It is not hard to see that the mean distortion is minimized when the quantizer function maps every input vector to its nearest codeword (or centroid), as shown in equation 2.15.

$$\phi_{KMeans}(x) = \text{argmin}_{c \in C}\, \texttt{dist}(x, c) = \text{argmin}_{c \in C} \|x - c\|^2 \tag{2.15}$$

While K-means quantization works decently, the amount of compression that can be achieved is not mindblowing. Since the entire codebook C needs to be stored in its original form in memory, for a codebook of size $|C| = K$, we use $O(K \cdot H)$ memory.

Product quantization

Product quantization (PQ) [36] is a way to make K-means quantization more memory efficient under the same codebook size constraint. This is achieved by leveraging a technique called *subspace division*, which involves breaking up the input vectors into subvectors of equal size. Product quantization can be seen as a foundational vector quantization algorithm since many other quantization algorithms originate from it [36, 53, 26, 4, 67]. Since product quantization represents an important part of the methodology used for this thesis, it is presented in greater mathematical depth in chapter 5. As a result of this, this subsection aims to provide the reader with only a basic intuition by illustrating how product quantization works when applied in the context of $M = 2$ subspaces for vectors of size $H = 4$.

Following up on the ideas mentioned above, let us consider our 4-dimensional input vector set X as described in equation 2.16.

$$X = \{ \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ x_{1,4} \end{bmatrix}, \begin{bmatrix} x_{2,1} \\ x_{2,2} \\ x_{2,3} \\ x_{2,4} \end{bmatrix}, \ldots, \begin{bmatrix} x_{n,1} \\ x_{n,2} \\ x_{n,3} \\ x_{n,4} \end{bmatrix} \} \tag{2.16}$$

Subspace division consists of splitting each of the 4-dimensional vectors into two 2-dimensional vectors, as shown in equation

$$\begin{aligned}
&\mathbf{x_i^{(1)}} := \begin{bmatrix} x_{i,1} \\ x_{i,2} \end{bmatrix} \; (i^{th} \text{ vector in subspace 1}) \\
&\mathbf{x_i^{(2)}} := \begin{bmatrix} x_{i,3} \\ x_{i,4} \end{bmatrix} \; (i^{th} \text{ vector in subspace 2}) \\
&X^{(s)} := \{\mathbf{x_1^{(s)}}, \mathbf{x_2^{(s)}}, ..., \mathbf{x_m^{(s)}}\} \text{ (input vectors in subspace s)}
\end{aligned} \tag{2.17}$$

The next step is to apply K-means quantization distinctly in each of the two vector subspaces corresponding to the two input vectors (training sets) $\{X^{(1)}, X^{(2)}\}$. Doing this will result in a pair of quantizers $(\phi^{(1)}, \phi^{(2)})$ and a pair of codebooks $(C^{(1)}, C^{(2)})$, containing the centroids obtained after applying clustering on the two 2-dimensional vectors sets $\{X^{(1)}, X^{(2)}\}$. The procedure is formally described in algorithm 1.

The global quantizer ϕ across the entire vector space is defined by applying quantization on each of the subspaces and then concatenating each of the outputs together to form a final quantized vector. The definition of the global quantizer for our two subspaces case is shown in equation 2.18.

Algorithm 1 PQ (s = 2, d = 4)

```
Require: $X = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_m}\} \subset \mathbb{R}^4$
  $X^{(1)}, X^{(2)} \leftarrow subspaceSplit(X)$
  $\phi^{(1)}, C^{(1)} \leftarrow kMeansQuant(X^{(1)})$
  $\phi^{(2)}, C^{(2)} \leftarrow kMeansQuant(X^{(2)})$
  return $(\phi^{(1)}, \phi^{(2)}), (C^{(1)}, C^{(2)})$
```

$$\phi(\mathbf{x}) = \begin{bmatrix} \phi^{(1)}(\mathbf{x^{(1)}}) \\ \phi^{(2)}(\mathbf{x^{(2)}}) \end{bmatrix} = \begin{bmatrix} \operatorname{argmin}_{\mathbf{c^{(1)}} \in C^{(1)}} \|\mathbf{x^{(1)}} - \mathbf{c^{(1)}}\|^2 \\ \operatorname{argmin}_{\mathbf{c^{(2)}} \in C^{(2)}} \|\mathbf{x^{(2)}} - \mathbf{c^{(2)}}\|^2 \end{bmatrix} \tag{2.18}$$

Since the codewords $c \in C$ that the global quantizer ϕ outputs are ultimately pairs of subcodewords $(c^{(1)}, c^{(2)})$ in the two subcodebooks $(C^{(1)}, C^{(2)})$, the codebook C is made up of all possible combinations of subcodewords. This concept can be mathematically expressed as a cartesian product, as shown in equation 2.19.

$$\begin{aligned} &C = C^{(1)} \times C^{(2)} \\ \\ &C - \text{the codebook for the whole vector space} \\ &C^{(1)} - \text{the subcodebook for the first subspace} \\ &C^{(2)} - \text{the subcodebook for the second subspace} \end{aligned} \tag{2.19}$$

From equation 2.19 and the fact that each subcodebook has the same cardinality $|C^{(1)}| = |C^{(2)}| = K$, it follows that the size of the global codebook C is $|C| = |C^{(1)}| \cdot |C^{(2)}| = K^2$. The power of *subspace division* and, subsequently, product quantization comes from the fact that, while our final codebook is much larger than for K-means quantization, our memory complexity remains exactly the same, i.e. $O(2 \cdot K \cdot \frac{H}{2}) = O(K \cdot H)$. An equivalent way of expressing this is that for a codebook of fixed size K, the memory complexity is $O(2 \cdot K^{\frac{1}{2}} \cdot \frac{H}{2}) = O(K^{\frac{1}{2}} \cdot H)$. The same idea can be generalized to an arbitrarily large number of subspace divisions M, in which case higher degrees of compressions are achieved, considering a fixed codebook size, as shown in table 2.2.

Quantization	M	Codebook cardinality	Memory complexity
K-means	-	K	$O(K \cdot H)$
PQ	2	K^2	$O(K \cdot H)$
PQ	m	K^m	$O(K \cdot H)$
K-means	-	K	$O(K \cdot H)$
PQ	2	K	$O(K^{\frac{1}{2}} \cdot H)$
PQ	m	K	$O(K^{\frac{1}{m}} \cdot H)$

Table 2.2: Comparison between codebook cardinality and memory complexity of PQ and K-means quantization.

Optimized product quantization

As the name suggests, optimized product quantization (OPQ) is an improvement over product quantization, which involves performing a *pre-rotation* to the input vectors before quantizing them. The concept was introduced by two different papers independently [26, 57], and it was motivated by the intuition that the *subspace division* is an arbitrary heuristic that might not be optimal for all datasets [53].

The idea behind optimized product quantization is to apply a rotation to the vector space before performing product quantization. From a mathematical point of view, a rotation is nothing more than a linear space transformation, and thus, it can be encoded as an orthogonal matrix R [26]. Our original problem is, hence, turned into an optimization problem with two objectives: finding an optimal rotation matrix R and an optimal quantizer-codebook pair (ϕ, C). The solution to the problem relies on the fact that given a fixed quantizer ϕ and codebook C, finding the optimal rotation matrix R is a mathematical problem with a known closed-form solution, namely the *Orthogonal Procrustes Problem* [26, 63]. As a result of this, the dual-optimization problem behind OPQ can be solved iteratively by alternating between finding an optimal (ϕ, C) given a fixed R and then finding an optimal R given a frozen (ϕ, C), as shown in algorithm 2.

Algorithm 2 Optimized product quantization

```
$R_0 \leftarrow I_H$(identity matrix of size H)
for $i$ in $[1, MaxIter]$ do
    $(\phi_i, C_i) \leftarrow PQ(R_{i-1}X)$
    $R_i \leftarrow OptimalProcrustesSolver(\phi_i, C_i)$
end for
return $(\phi_{MaxIter}, C_{MaxIter}, R_{MaxIter})$
```

If, in the case of PQ, we would only be saving the codebook in memory as a required artefact of the compression [36, 53], in the case of OPQ, we need to store both the codebook C and the rotation matrix R [26]. Since the size of the rotation matrix R is H^2, it follows that, excluding the stored index assignments, the memory complexity of OPQ becomes $O(K^{\frac{1}{m}} \cdot H + \cdot H^2) = O((K^{\frac{1}{m}} + 1) \cdot H) = O(K^{\frac{1}{m}} \cdot H)$. In other words, the memory complexity remains unchanged.

As a result of the above, the main takeaway that we can get from OPQ is that the performance of a vector quantization algorithm can be improved by *employing rotations* at the cost of an insignificant memory increase, which does not have an impact on asymptotic complexity.

Another important observation is that PQ is a submethod of OPQ. Hence, by creating an improved quantization algorithm that is functionally equivalent to PQ we are automatically improving OPQ as well.

2.3.3. Supervised quantization

In the previous subsections, we have gone over the classical formulation of vector quantization, and we have shown that under this formulation, product quantization is essentially an unsupervised learning algorithm whose objective is to minimize the mean distortion, as expressed by equation 2.13. We have also seen examples of several concrete implementations of vector quantization, which illustrate how techniques such as *subspace divisions* or *space rotations* can be used to improve memory complexity and performance.

Taking all of the above into account, the main question that arises is: what if we want to apply vector quantization to problems for which the minimization of mean distortion (equation 2.13) is not the ultimate goal? It turns out that achieving this requires a reformulation of the classical unsupervised vector quantization framework, which opens the way to a novel category of the quantization algorithms, namely the *supervised quantization algorithms* [53, 13, 22, 72, 70]. While several such algorithms have been developed in the last few years, we will go over three key ones [13, 22, 72].

DPQ

DPQ [13] is one of the first models that tried to address the limitations of PQ and its derivatives when it comes to transitioning from an unsupervised to a supervised problem setting. Since many of the previous quantization algorithms involved non-differentiable operations [53, 36, 26], training with respect to an arbitrary loss function would be difficult. This limitation is tackled by DPQ, which introduces a framework for performing differentiable product quantization.

DPQ introduces a reverse-quantizer (called *reverse discretization function* in the paper), which leads to a re-definition of the original mathematical quantization formulation. The neural architecture of DPQ follows the design pattern of an encoder-decoder network [74]. The quantizer, which becomes a learnable encoder, creates a mapping from the input space to the codebook ($\phi : \mathbb{R}^H \rightarrow C$), while the reverse quantizer maps the codebook back to a continuous vector containing the quantized vector outputs ($\rho : C \rightarrow \mathbb{R}^H$). Both the quantizer $\phi(\cdot; K)$ and the reverse quantizer $\rho(\cdot; V)$ are learnable differentiable functions. Their learnable parameters are contained in the matrices K and V, referred to as the *key matrix* and the *value matrix*.

DPQ is a flexible quantization model that could be trained in either an *unsupervised* or a *supervised* setting. In an unsupervised learning context, DPQ would be trained like an auto-encoder to minimize the difference between the original vector x and its quantized counterpart $\rho(\phi(x))$. However, as a consequence of all operations being differentiable, any arbitrary loss function $L(\rho(\phi(x)), y)$ could be used for training, enabling *supervised learning*.

JPQ

A more effective alternative to DPQ [13] is JPQ [22]. JPQ takes a different approach from DPQ, an approach which involves training the quantizer and query encoder together. This way of training leads to better ranking performance but at the cost of higher training requirements. The main techniques that lie at the core of JPQ are:

- *ranking-oriented loss*
- *PQ centroid optimization*
- *end-to-end negative sampling*

As the name suggests, the main idea behind ranking-oriented loss is the use of a loss function that penalizes poor ranking performance. Hence, in JPQ, the quantizer and the query encoder learn to adapt to each other to deliver optimal document or passage ranking results.

Differentiability in JPQ is achieved by means of *PQ centroid optimization*, which is a different way of obtaining differentiability than the one used in DPQ. Unlike DPQ, JPQ starts by applying PQ on the input embeddings, yielding a codebook C and a set of vector-to-codeword assignments induced by the quantizer function ϕ. During supervised training, JPQ adjusts the codewords created by PQ but keeps the index assignments frozen.

During the training process, JPQ performs end-to-end negative sampling. This means that rather than pre-computing the positive-negative pairs used during training, these pairs are obtained in real time by performing retrieval as the model learns. This results in a training process that is closer to the behaviour of the real time system and, hence, more efficient.

RepCONC

Similarly to JPQ [22], RepCONC [72] is a supervised quantization method that aims to adapt the quantizer and the dual-encoder together for the purpose of increased ranking performance. However, while JPQ only trains the query encoder, RepCONC involves training the full dual encoder together with the quantizer.

RepCONC achieves the training of the dual-encoder and PQ together [36] by leveraging a technique called *constrained clustering*. Like JPQ, RepCONC relies on the *ranking-oriented loss* to obtain supervised signals that iteratively improve the re-ranking performance of the system. The authors of RepCONC argue that the design decision made in JPQ to keep the index assignments fixed leads to suboptimal results. Since, according to them, relying exclusively on the ranking-oriented loss would lead to unexpected behaviour in the context of learnable index assignments, RepCONC uses a combination of the supervised ranking-oriented loss and the unsupervised distortion (MSE), as indicated by equation 2.20.

$$\begin{aligned}
&L := L_r + \lambda L_m \\
&L_m := \|\eta_d(d) - \hat{\eta}_d(d)\|^2 \\
&L_r := -\log \tfrac{\exp(\eta_q(q) \cdot \hat{\eta}_d(d^+))}{\hat{\eta}_d(d^+) + \sum_{d^-} \exp(\eta_q(q) \cdot \hat{\eta}_d(d^-))} \\
&\\
&L - \text{loss function} \\
&L_m - \text{distortion (MSE) loss} \\
&L_r - \text{ranking oriented loss} \\
&\lambda - \text{parameter controlling the trade-off between the two losses}
\end{aligned} \tag{2.20}$$

Despite being used together, the ranking-oriented loss and MSE have divergent objectives. While one of them encourages performing representation learning in a way that guarantees the distinctiveness of representations, the other one forces the system to learn overlapping representations of documents. The issue is solved by *uniform clustering*. This means that documents are assigned uniformly across the clusters (i.e. each cluster has an approximately equal number of documents). It can be proven that uniform clustering minimized the degree of representation overlap, reducing the degree of divergence between the two learning objectives.

3 Problem setup

This chapter aims to provide the reader with a formal introduction to the problem we are solving and a brief overview of the contributions (in relation to the research questions) that lie at the heart of the thesis. To this end, the first section, entitled *Problem setup*, lays the theoretical basis of the thesis by introducing the context of the research in its full mathematical rigour. Building on the first section, the second section defines the general problem that the thesis tackles, which acts as a foundation for the subsequent research questions. Lastly, the third section provides a brief overview of the contributions that this thesis has made to information retrieval.

3.1. Problem setup

3.1.1. Algorithmic formulation of two-stage retrieval

As the reader might have seen in the previous sections of the thesis, two-stage retrieval systems, which constitute the focus of our research, follow a well-defined algorithmic framework, which is described in the following paragraphs.

Our setup starts with a corpus of documents (or passages, which is equivalent) $D = \{d_1, d_2, ..., d_n\}$ and a query q. The query q is assumed to be sampled from some probability distribution $F_Q : Q \to [0, 1]$ that captures general user behaviour. Both the corpus D and the query q are textual data. The corpus D is assumed to be static and known before runtime. In contrast, the query q is only received at runtime when the user interacts with the information retrieval system. The purpose of the two-stage retrieval system is to accurately retrieve the most relevant documents with respect to the query and order them by their relevance. The described setup is illustrated by figure 3.1.

As its name suggests, a two-stage retrieval system is made up of two functional components. The first component is the retriever, whose aim is to find a subset of documents in the corpus that are likely relevant for the query [71, 44]. We will refer to this subset as the first stage retrieval set S_k^q for the query q, where k is the size of the set ($k := |S_k^q|$). We can mathematically model the behaviour of the retriever as a function defined on all possible queries, parameterized by the corpus D and the size of the retrieval set k. The retriever function returns a retrieval set S_k^q for every possible user query q, as described by equation 3.1.

$$\begin{aligned}
&Retriever_{D,k} : Q \to P(D) \\
&Retriever_{D,k}(q) := S_k^q \subseteq D \\
\\
&Q := \text{the set of all possible queries} \\
&P(D) := \text{the power set of the corpus } D \\
&S_k^q := \text{the first stage retrieval set of the query q}
\end{aligned} \tag{3.1}$$

Since most retrievers create the retrieval set S_k^q by assigning some relevancy score to the documents in the corpus, we will formally define a function that provides the first stage retrieval score of a document

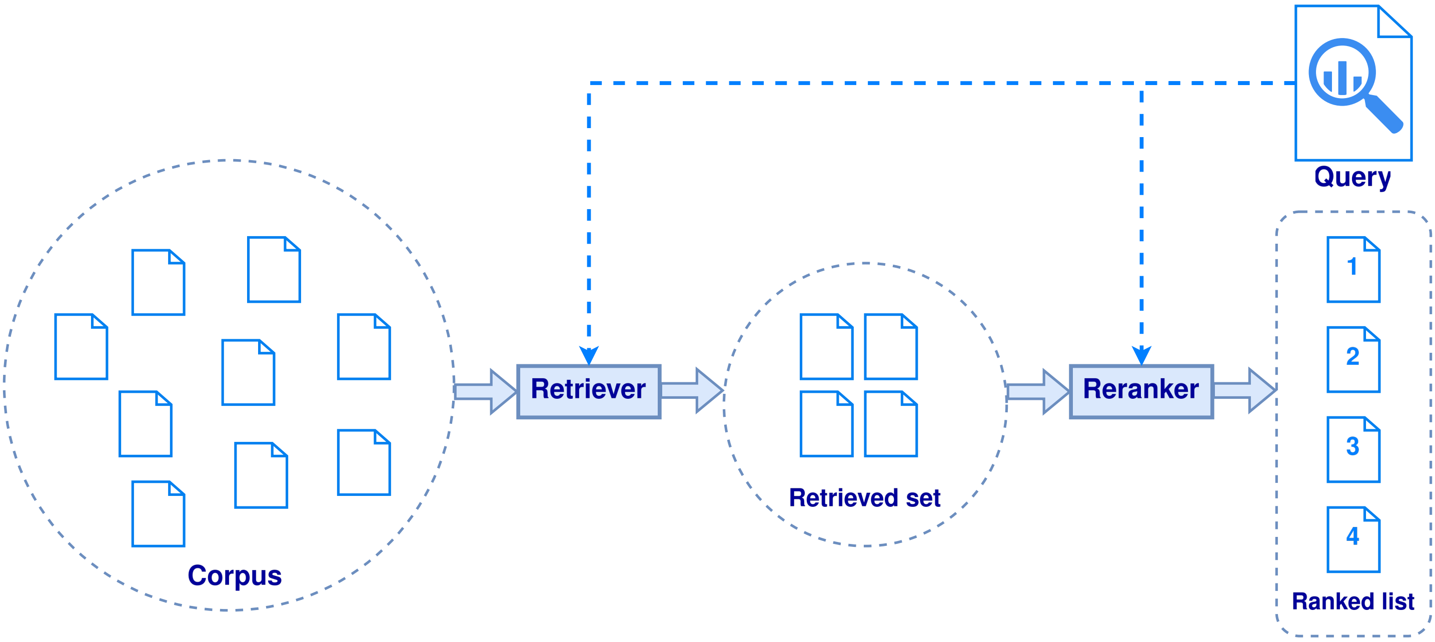


Figure 3.1: Diagram describing the general problem setup. The retriever selects a subset of relevant documents from the corpus based on the query. The re-ranker orders the documents by their relevancy.

with respect to the query. The first stage relevancy score function is described by equation 3.2.

$$\begin{aligned}&retrievalScore : D \times Q \to \mathbb{R}\\&retrievalScore(d, q) := \text{numerical value expressing the relevancy of document } d \text{ for } q\end{aligned} \tag{3.2}$$

The second component of a two-stage retrieval system is the re-ranker. The role of the re-ranker is to order the documents in the first stage retrieval set S_k^q according to their query relevancy. Similarly to the retriever, we will formally define the re-ranker as a function that takes the retrieval set and the query as input and whose output is a sequence of integer numbers representing the rank (or order) of each document in the set. This definition is provided by equation 3.3.

$$\begin{aligned}&Reranker_{D,k} : P(D) \times Q \to \mathbb{Z}^k\\&Reranker_{D,k}(S_k^q, q) := \mathbf{r_q} = [r_{q,1}, r_{q,2}, ..., r_{q,k}]\\&\\&Q := \text{the set of all possible queries}\\&P(D) := \text{the power set of the corpus } D\\&S_k^q := \text{the first stage retrieval set of the query q}\\&\mathbb{Z}^k := \text{the set of all integer vectors of size k}\\&\mathbf{r_q} := \text{vector encoding the rank (order) for all retrieved documents}\\&r_{q,i} := \text{integer representing the rank (order) of the ith document in the set}\end{aligned} \tag{3.3}$$

In the same way as the retriever, the re-ranker will order the retrieved documents by assigning relevancy scores. The main difference between the relevancy score of the retriever and the re-ranker is that the latter is assumed to be more accurate but slower to compute. Hence, the re-ranking score will follow the exact formal definition as the retrieval score, as shown in equation 3.4.

$$\begin{aligned}&rerankingScore : D \times Q \to \mathbb{R}\\&rerankingScore(d, q) := \text{numerical value expressing the relevancy of document } d \text{ for } q\end{aligned} \tag{3.4}$$

Having defined all of the above, we can formulate a generic algorithmic framework that captures the behaviour of a two-stage retrieval system. Our framework contains two subprocedures that can be closely mapped to the two functional components of the system, the retriever and the re-ranker. The main procedure of the framework combines the two subprocedures to achieve the desired behaviour, i.e. an efficient search engine. The abstract formulation of the algorithmic framework can be found in Algorithm 3.

Algorithm 3 Abstract formulation of two-stage retrieval

```
procedure Retriever($D, k, q$)
    $S_k^q \leftarrow \{\}$
    for $d$ in $D$ do
        if len($S_k^q$) $< k$ then
            $S_k^q \leftarrow S_k^q \cup \{(\texttt{retrievalScore}(d, q), d)\}$
        else
            $minScore \leftarrow \min(S_k^q)[0]$
            $currentScore \leftarrow \texttt{retrievalScore}(d, q)$
            if $currentScore > minScore$ then
                $S_k^q \leftarrow S_k^q \cup \{(currentScore, d)\} \setminus \{\min(S_k^q)\}$
            end if
        end if
    end for
    return $S_k^q$
end procedure

procedure Reranker($S_k^q, q$)
    $rankingScoreList \leftarrow []$
    for $(retScore, d)$ in $S_k^q$ do
        $rankingScoreList \leftarrow \texttt{append}(rankingScoreList, \texttt{rerankingScore}(d, q))$
    end for
    $r \leftarrow \texttt{argSort}(rankingScoreList)$
    return $r$
end procedure

procedure TwoStageRetrieval(D, k, q)
    $S_k^q \leftarrow \texttt{Retriever}(D, k, q)$
    $r \leftarrow \texttt{Reranker}(S_k^q, q)$
    return $S_k^q, r$
end procedure
```

3.1.2. Sparse retrieval

Having seen an abstract, generic formulation of the two-stage retrieval algorithm, it is time to have a closer look at the first component of the system, the retriever. More specifically, we will look at how the $retrievalScore$ function (equation 3.2) is defined in the context of our setup.

As previously mentioned, the role of the $retrievalScore$ function is to assign a relevancy score to a given document-query pair, represented as textual (character string) data. In the theoretical background section of the thesis, we have presented multiple families of approaches that can be employed to achieve this objective. While modern approaches make use of data-driven, deep-learning-based, natural language processing techniques [40, 56, 47, 46, 71, 44], we have seen that such techniques come at the disadvantage of high computational time, which leads to slow retrieval in situations where the document corpus D is large. Since the core idea of the two-stage retrieval pipeline is to perform a fast, lower accuracy retrieval step and improve the accuracy in the re-ranking stage [44], a better design choice is to perform *sparse retrieval*.

Sparse retrieval is an umbrella term for a family of pre-deep-learning era information retrieval techniques that became popular and widely used in industry for their relatively high accuracy and low computational time [71, 61, 73]. The core assumption behind the sparse retrieval models is to look at treat both documents and queries as *lexical bags* [61, 3]. This means that a sparse retrieval system will model both documents and queries as unordered bags of words, where each word is an atomic element. Hence, the relevancy score will be calculated on the basis of bag statistics that, ultimately, represent a measure of word (or lexical) overlap between the query and the document. The exact formula that computes the relevancy score has its foundations in probability and information theory [3].

The sparse retrieval method that was chosen for the setup analyzed in this thesis is *BM25* [61], and its formula is defined in equation 3.5. BM25 is one of the most popular sparse retrieval scoring functions. Since BM25 is still one of the most widely used benchmarks in information retrieval research [47, 70, 46, 44, 40], it made sense for it to become part of our setup, as it made our results more comparable to previous research.

$$
\begin{array}{l}
retrievalScore(d,q) := bm25(d,q) = \sum_{w \in q} \frac{f_{w,d} \cdot (k_1+1)}{f_{w,d} + k_1 \cdot (1 - b + b \cdot \frac{|d|}{\texttt{avgdl}})} \cdot \log \frac{N - n_w + 0.5}{n_w + 0.5} \\
f_{w,d} := \text{the frequency of word } w \text{ in document } d \\
n_w := \text{number of documents in the corpus containing word } w \\
N := \text{number of documents in the corpus} \\
|d| := \text{the length of document } d \\
\texttt{avgdl} := \text{the average length of a document in the corpus} \\
b := \text{paramater controlling the impact of document length on word (term) saturation} \\
k_1 := \text{parameter controlling word (term) staturation}
\end{array}
\tag{3.5}
$$

BM25 is an improved version of TF-IDF [3]. The core idea behind BM25 is to iterate over all words in the input query. A document d will get a higher score if it contains many occurrences of the words in the query, which corresponds to the $\frac{f_{w,d} \cdot (k_1+1)}{f_{w,d} + k_1 \cdot (1 - b + b \cdot \frac{|d|}{\texttt{avgdl}})}$ factor of the formula. However, since certain words are generally more likely to occur than others, the score provided by each word is weighted down by the frequency of the word across the entire corpus, corresponding to the $\log \frac{N - n_w + 0.5}{n_w + 0.5}$ factor.

An improvement that BM25 makes over TF-IDF is the introduction of term saturation. The intuition behind term saturation is that the more a particular word (term) occurs in a document, the less each new occurrence of the word will contribute to the relevancy. In other words, if a specific document contains 1000 occurrences of the word *duck* and another document contains 2000 such occurrences, it is presumably incorrect to claim that the second document is twice more relevant than the first. This is because both documents contain enough *ducks* to be assumed as very relevant for the informational need corresponding to the beloved winged animal. How fast terms (words) saturate and how much the document length influences the saturation speed are controlled by the parameters k_1 and b.

3.1.3. Dense re-ranking

The previous subsection provided an overview of sparse retrieval, the technique chosen for our system's retriever component. We have seen that the fundamental assumption behind sparse retrieval is the bag-of-words model of the query and document. The fact that the bag-of-words model treats terms as atomic elements is why sparse retrieval is very fast [61, 71]. While the *term atomicity* can be seen as one of the main strengths of the technique, it is, at the same time, one of its most significant weaknesses. The reason for this is the fact that the term atomicity prevents the retriever model from handling synonymous words, as they will be represented as distinct, unrelated elements. The inability of sparse retrieval systems to deal with synonyms is formally known as the *vocabulary mismatch problem* [25, 75, 30], and it is the main reason behind why dense, data-driven, relevancy scoring techniques were developed.

Taking into account that the role of the re-ranker is to tackle the deficiencies induced by having used a sparse retrieval in the first stage of the system, the reason why a dense model has been chosen as the relevancy scoring function behind the re-ranker becomes obvious. As the reader has seen in the theoretical background section, many categories of *dense* methods have been developed over the years [56, 40, 71]. However, the state-of-the-art considering all characteristics required for information retrieval, which also constitutes the focus of our research, is the *dual-encoder* [37, 71].

The dual-encoder is a relevancy scoring technique that heavily relies on an area of deep learning known as representation learning [6]. As its name suggests, the dual-encoder involves the training and use of two distinct transformer-based neural networks [66, 39] responsible for encoding the documents and queries in a shared vector space [40, 47, 46, 71]. The characteristics of the vector space differ from dual-encoder to dual-encoder. However, across the different techniques, the vector space shares two characteristics:

- the vectors encode the semantics of the document or query
- distances between vectors in the space correspond to semantic closeness behind pieces of text (document or query)

As mentioned in the general problem setup, the corpus of documents D over which we search is known before runtime. This fact enables the dual-encoder to pre-compute the vector encoding for all the documents in the corpus, which is a way of trading memory for speed [37, 40, 71]. This also means that the only computations that the dual-encoder needs to perform during runtime are encoding the query and then computing the distances between the query and the document encodings. The general workings of a dual-encoder are illustrated by figures 3.2, 3.3 and 3.4.

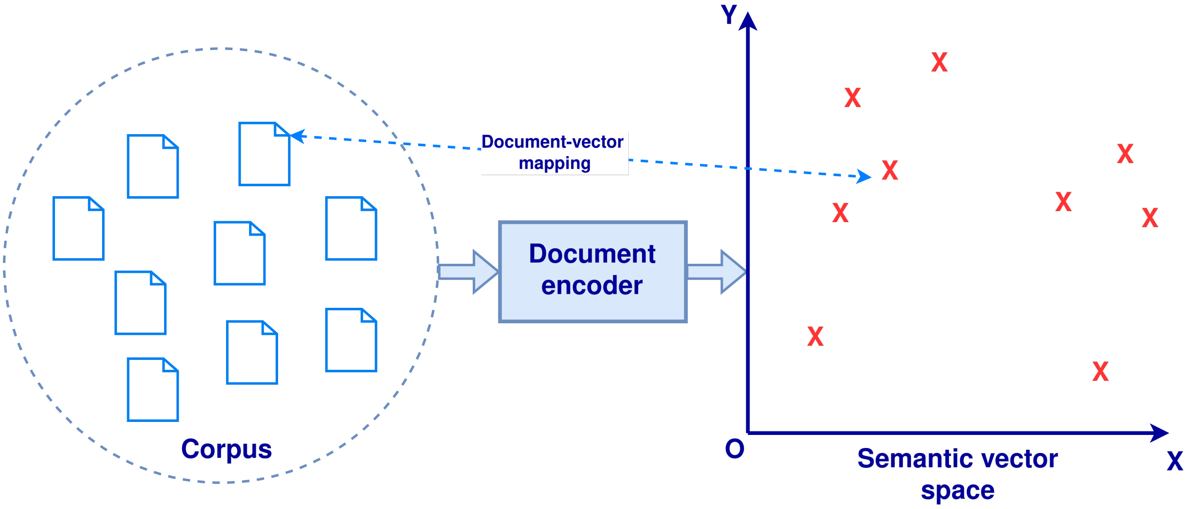


Figure 3.2: Diagram illustrating the encoding of the document corpus into a semantic vector space by the document encoder of a generic dual-encoder. As mentioned in this chapter, the encoding of the document corpus is performed before runtime and the vector encodings are stored in memory.

It was previously mentioned that the application of quantization on ColBERT [40], a very popular BERT-based dual-encoder, has already been explored. The paper's authors introduce a new type of quantizer, called **CQ** (contextual quantizer), which relies on both context-free and contextual BERT encodings [70]. The latter design choice makes the proposed quantizer dependent on dual-encoders that produce token-level embeddings, such as ColBERT. As a result, the application of quantization on single-vector dual-encoders, which by design produces smaller semantic vector spaces, is unexplored in the context of re-ranking. Since quantization's primary purpose is reducing the vector space's size, as we have seen in the theoretical background, exploring quantization on more memory-efficient dual encoders is a logical next step. This is why single-vector dual encoders have been chosen for the re-ranker component.

The specific single-vector dual-encoders chosen for analysis are TCT-ColBERT [47] and Aggretriever [46]. While TCT-ColBERT is obtained by distilling ColBERT into a dual-encoder that computes similarity by dot-product, Aggretriever consists of a learned aggregation of the token embeddings into a single vector.

Regardless of the chosen single-vector dual-encoder, the re-ranker follows the same algorithmic flow. The dual-encoder comprises two functional components, a document encoder and a query encoder, that can be mathematically modelled as shown in equation 3.6.

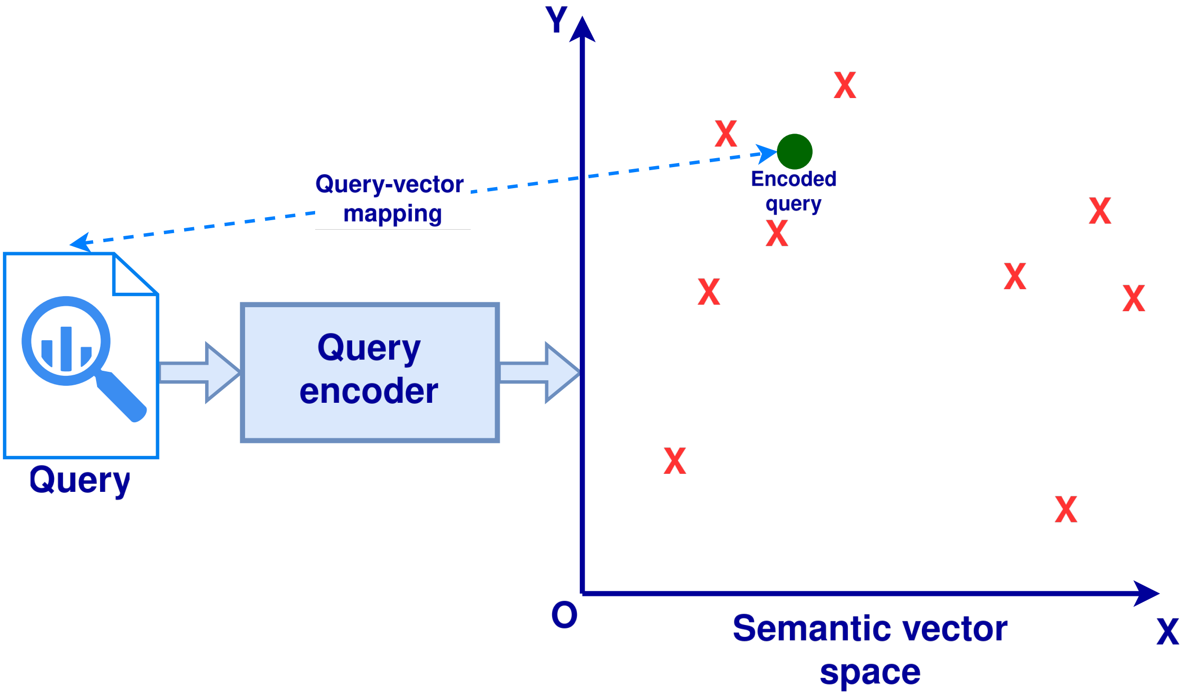


Figure 3.3: Diagram illustrating the encoding of a query into the semantic vectory space by the query encoder of a generic dual-encoder. As mentioned, the encoding of the query is performed during runtime, in a real-time fashion.

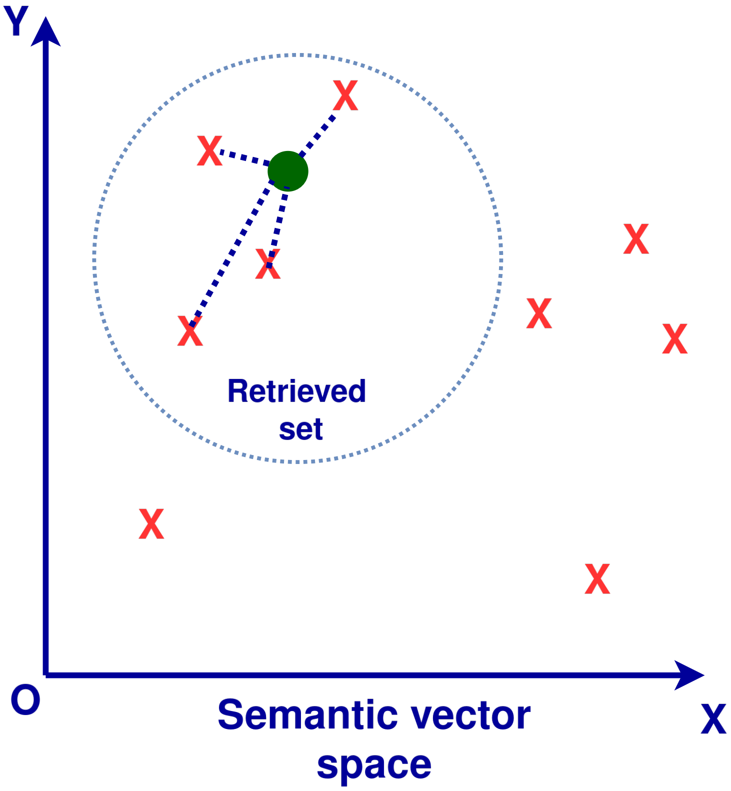


Figure 3.4: Diagram illustrating the distance computation between the encoder query and the encoded documents from the retrieved set.

$$
\begin{aligned}
&\eta_q : Q \to \mathbb{R}^h \\
&\eta_q(q) := \text{the encoding of query } q \\
&\eta_d : D \to \mathbb{R}^h \\
&\eta_d(d) := \text{the encoding of document } d
\end{aligned}
\tag{3.6}
$$

Since both TCT-ColBERT and Aggretriever use the *dot product* as their distance metric for their semantic space, the re-ranking score in the context of our system is defined as the dot product between the query encoding and the document encoding, as indicated by equation 3.7.

$$rerankingScore(d, q) := \eta_d(d) \cdot \eta_q(q) \tag{3.7}$$

3.2. Problem definition

In the previous section, we have gone over the technical details behind the system we are analyzing: a two-stage retrieval system composed of a sparse first-stage retriever and a dense single-vector dual-encoder re-ranker. As mentioned in the introduction, this thesis aims to **reduce the memory used by such a system**.

It is not hard to see that the largest amount of memory in the two-stage retrieval pipeline is the *pre-computed dense index used by the dual-encoder*. The reason for this is that the document corpus that a retrieval system needs to work with can get arbitrarily large. As an example, MSMARCO Passage, a dataset commonly used by information retrieval researchers, has roughly 8.8 million passages [5, 11]. A dual-encoder with a small memory footprint, like TCT-ColBERT, outputs a single vector with 768 dimensions per passage [47]. Assuming that each numerical value is stored as a 32-bit floating point number, the resulting dense index for the MSMARCO Passage dataset would have roughly 27.2 GB, which is beyond the capabilities of most commodity hardware to hold in the RAM memory. Since, ideally, we would like to scale the retrieval pipeline to larger and larger corpora, it becomes clear why tackling the memory issues induced by the use of dual-encoder re-rankers is of high importance.

Following up on the ideas of the previous paragraph, our problem can be formally thought of as compressing the size of a collection of vectors (i.e. the document embeddings created by the dual-encoder in our setup) in a way that the performance of the downstream task (the re-ranking of documents with respect to some query) is hurt as little as possible. In other words, we are looking into the use or development of a vector compression algorithm that can achieve a good trade-off between memory compression and the deterioration of ranking performance. An illustration of the problem being tackled is provided in figure 3.5.

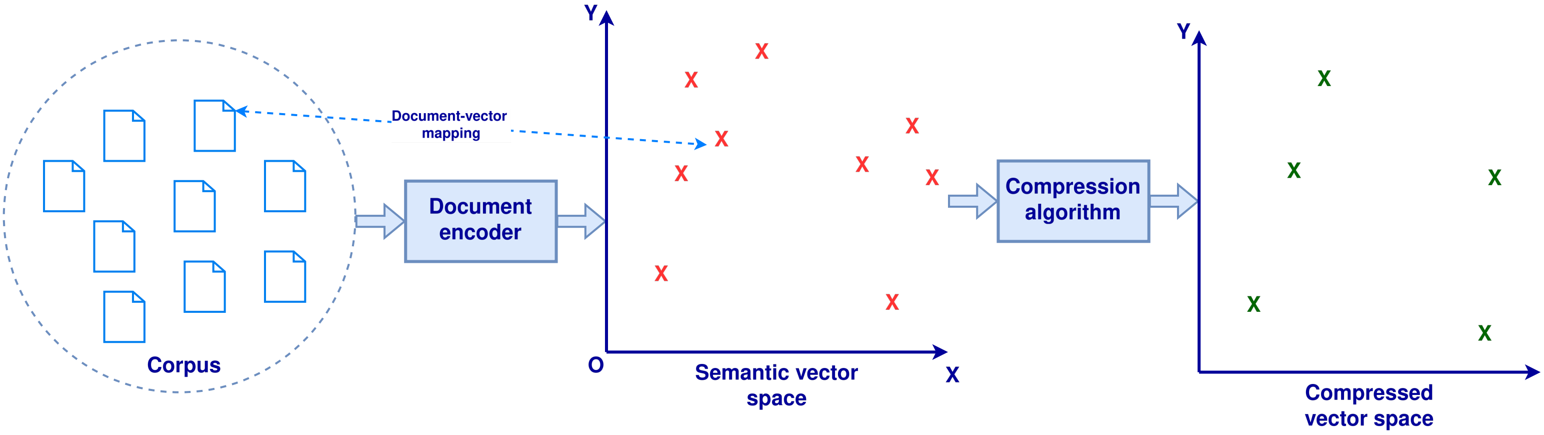


Figure 3.5: Diagram illustrating the idea of compressing the vector space composed of the document encodings.

While many families of compression algorithms could be applied, one promising option, as indicated by its good performance in the context of dense retrieval (or approximate nearest neighbour search), is vector quantization [53].

As we have seen in the theoretical background section, from a mathematical perspective, vector quantization can be seen as an algorithm that finds a mapping between a continuous vector space and a discrete vector space in such a way that the distance between the original vector and its discrete space representation is minimized. One of the most basic forms of vector quantization would be using K-means clustering on the original vector space and then representing each vector in the cluster as the cluster's centroid [53, 36, 1]. Doing this would, of course, lead to many neighbouring vectors being reduced to the same vector, making the re-ranker unable to distinguish between them and, thus, hurting the re-ranking performance.

We will formally introduce vector quantization to our general problem setup by defining a quantizer function ϕ that maps any vector in the semantic vector space to its quantized representation in the compressed space, as indicated by equation 3.8.

$$\begin{aligned}
&\phi : \mathbb{R}^h \rightarrow C \\
&\phi(\eta_d(d)) = \hat{\eta}_d(d) \in C \\
&\phi := \text{the quantizer function} \\
&C := \text{the codebook associated to the quantizer} \\
&\eta_d(d) := \text{the vector embedding associated to document d} \\
&\hat{\eta}_d(d) := \text{the quantized vector embedding associated to document d}
\end{aligned} \tag{3.8}$$

Introducing vector quantization to the two-stage retrieval pipeline leads to a slight modification of the `rerankingScore` function defined in equation 3.7. Namely, rather than computing the re-ranking score using the original document encoding, we use the quantized document encoding instead, as shown in equation 3.9.

$$rerankingScore(d, q) := \phi(\eta_d(d)) \cdot \eta_q(q) = \hat{\eta}_d(d) \cdot \eta_q(q) \tag{3.9}$$

Having formally added vector quantization to our problem setup leads to a more precisely formulated definition of our general problem. **Can we find a quantizer function ϕ and codebook C that minimizes the memory footprint of the dual-encoder while preserving as much of its re-ranking performance as possible?**

3.3. Cotributions

In the previous section, we formally defined the problem we aim to solve, i.e., finding an optimal quantization algorithm for dual-encoder-based re-ranking. Since the problem is extensive and completely unexplored (as far as we are aware) in the context of single-vector dual-encoders [70], this thesis aims to make the first steps in its direction, laying the groundwork for future research. To this end, the scope of this thesis is to answer the research questions by bringing the following contributions:

- studying the effectiveness of product quantization on single-vector dual-encoders in the context of passage re-ranking.
- investigating if, by leveraging sparse score interpolation, higher level of compression with similar performance decay can be obtained.
- proposing and evaluating **WolfPQ**, a learnable quantization method aimed at further improving quantization for re-ranking by bridging the gap between the objective functions used in training product quantization and re-ranking systems respectively.

4
Theoretical foundation

In the previous chapter, we formally introduced the problem and provided the reader with an overview of the research questions. Following up on this, this chapter will present the reader with a summary of a few papers that are intimately related to the research focus of this thesis. Hence, the first chapter will go over a paper entitled *Fast Forward Indexes for Efficient Document Ranking* [44], from which our research borrows the problem setup, as well as the idea of re-ranking with interpolation. The second section will provide an overview of *Jointly Optimizing Query Encoder and Product Quantization to Improve Retrieval Performance* [22], which introduces a state-of-the-art supervised quantizer for dense retrieval. Finally, the last section will present *Compact Token Representations with Contextual Quantization for Efficient Document Re-ranking* [70], which is the only paper that tackles similar research questions as us and whose limitations are addressed in this thesis.

4.1. Fast Forward Indexes

The paper *Fast Forward Indexes for Efficient Document Ranking* [44] was born out of the necessity to reduce the query processing times of neural transformer-based passage re-ranking methods. While such methods have been proven numerous times to produce gains in ranking performance, their use in practice was limited by an associated increase in online processing time. According to the paper, the high query processing times are a direct result of the use of over-parameterized cross-attention models for scoring the retrieved passages.

The paper addresses these limitations by introducing a concept called *fast forward indexes*. The idea of fast forward indexes is to replace the above-mentioned cross-attention models with dual-encoders that allow for the separation between document (passage) and query processing. Relying on the insight that interpolation-based re-ranking is a computationally inexpensive way to improve re-ranking performance, the paper proposes a re-ranking setup in which sparse retrieval is used to retrieve small document subsets, which are re-ranked using their *dual-encoder scores* linearly interpolated with their *retrieval sparse scores*. As the fast forward indexes setup has proven to be a cost-efficient way to perform highly accurate re-ranking, we ended up adopting it as the framework under which the quantization research of this thesis is performed. The setup corresponding to the fast forward indexes framework has already been presented in great detail in the previous chapter.

Another concept introduced by the authors of the paper is *sequential coalescing*, an optimization technique aimed to reduce the memory of the dual-encoder dense index, which the previous chapter has shown to be a significant bottleneck in the re-ranking framework of our study. The main idea behind sequential coalescing is to group together passages whose semantic vectors are close to each other and represent them by the average of the group, as indicated by figure 4.1. The threshold-based grouping of vectors used in *sequential coalescing* can be seen as a form of vector clustering, which in the theoretical background chapter has been shown to be a basic form of quantization. Hence, our research into vector quantization for the dual-encoder-sparse-interpolated re-ranking setup is an attempt to improve *sequential coalescing*, which the paper proves to be a promising technique to reduce the memory requirements, at the cost of only slight deteriorations to the re-ranking performance.

In light of all the above, it is not hard to see that this thesis can be framed as a direct continuation of *Fast Forward Indexes for Efficient Document Ranking*, having the aim to explore ways of reducing the memory usage of the dual-encoder based re-ranking setup even further.

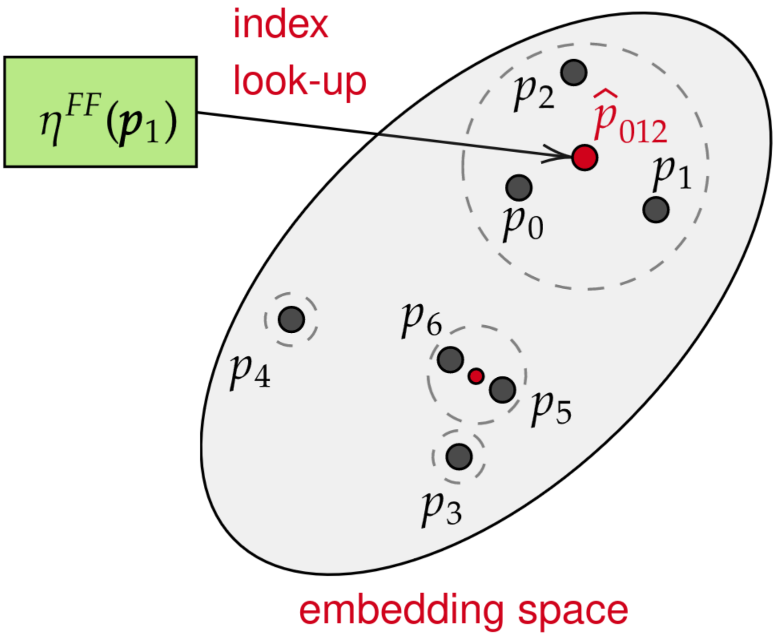


Figure 4.1: Diagram illustrating sequential coalescing. Figure taken from [44]

4.2. JPQ

Jointly Optimizing Query Encoder and Product Quantization to Improve Retrieval Performance [22], commonly abbreviated as JPQ, is a popular paper in the vector quantization research community that introduces a supervised quantization algorithm for dense retrieval. The main motivation behind the paper is the necessity to reduce the size of the indices used in dense retrieval. As the authors of the paper mention, while previous techniques of performing vector compression already exist, these techniques suffer from a series of limitations. On the one hand, as we also pointed out in the previous sections of the thesis, many of the compression methods do not benefit from supervised signals during training, as they are trained by optimizing the reconstruction error, which can be seen as an unsupervised form of training. In addition to this, the document encoders and the quantizer algorithms are often trained separately, which can lead to incompatibilities and decreased performance. Finally, while some studies into jointly training the document encoder and quantizer already exist, they do not focus on web search, for which negative sampling-based training is of high importance. The authors of the paper tackle all of these limitations by proposing a novel quantization algorithm called *JPQ*.

From an architectural perspective, JPQ can be seen as a trainable extension of PQ [36], which follows the training workflow indicated by figure 4.2. As shown in the figure, JPQ is trained in an end-to-end fashion using a loss function that takes into account the quality of the ranking results. The training process involves iterating over all queries in the training dataset, performing negative sampling using those queries, quantizing the positive and negative document embeddings, computing the dense ranking scores using these embeddings and finally ranking the documents based on these scores to calculate the ranking-oriented loss of the model.

The core techniques that lie at the heart of JPQ are the use of the *Ranking-oriented loss*, *PQ centroid optimization*, and *End-to-End negative sampling*.

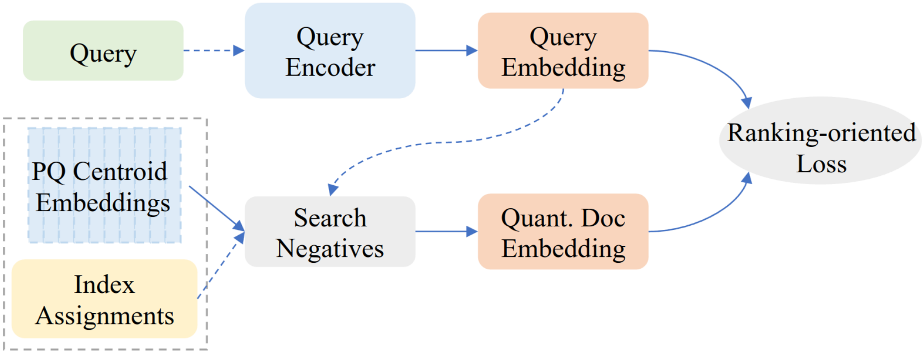


Figure 4.2: Training workflow of JPQ. In the diagram, backpropagated gradients are indicated by solid lines. Figure taken from paper [22].

The ranking-oriented loss proposed by JPQ is a pairwise loss that compares the dense scores of pairs of positive and negative document samples obtained after quantization, as indicated by equation 4.1. While the paper only specifies the form, but not the exact formula used by the loss, the authors suggest that hinge loss [16], RankNet [8] and LambdaRank [9] are all viable options.

$s^{\dagger}(q,d)$ – the dense score computed based on query q and the quantized embedding of document d
$l(s^{\dagger}(q,d^{+}),s^{\dagger}(q,d^{-}))$ – the form of the ranking-oriented loss function
(4.1)

The idea of PQ centroid optimization is to initialize the codebook of JPQ with the centroids resulting from training PQ on the target document dataset. During the training process, the PQ centroids are treated as learnable parameters and are further optimized using backpropagation based on the gradients shown in equation 4.2. The authors of the paper choose to keep the index assignments frozen, as their claim is that it is challenging to create differentiable approximations for index assignments.

$$\frac{\delta l(s^{\dagger}(q,d^{+}),s^{\dagger}(q,d^{-}))}{\delta c_j^{(i)}} = \begin{cases} -\alpha q^{(i)} & j = \phi^{(i)}(d^{+}), j \neq \phi^{(i)}(d^{-}) \\ \alpha q^{(i)} & j \neq \phi^{(i)}(d^{+}), j = \phi^{(i)}(d^{-}) \\ 0 & j = \phi^{(i)}(d^{+}), j = \phi^{(i)}(d^{-}) \\ 0 & j \neq \phi^{(i)}(d^{+}), j \neq \phi^{(i)}(d^{-}) \end{cases}$$
$$\alpha = \frac{\delta l(s^{\dagger}(q,d^{+}),s^{\dagger}(q,d^{-}))}{\delta s^{\dagger}(q,d^{-})} \qquad (4.2)$$

$\frac{\delta l(s^{\dagger}(q,d^{+}),s^{\dagger}(q,d^{-}))}{\delta c_j^{(i)}}$ – gradient of the loss function with respect to the subcodebook $c_j^{(i)}$
$q^{(i)}$ – ith subspace division of the query embedding
$\phi^{(i)}$ – subquantizer corresponding to the ith subspace

Last but not least, JPQ relies on end-to-end negative sampling, which has been proven to be useful for training dual-encoders. The intuition behind negative sampling is to penalize top-ranked negatives, while ignoring low-ranked documents that are truncated by the evaluation function regardless.

The sampling of top-ranked negative documents is performed in real-time by performing dense retrieval based on the current version of the PQ centroid embeddings, as indicated by equation 4.3.

$$D_q^{-\dagger} = \texttt{sort}(d \in C \setminus D_q^{+}\text{ordered by } s^{\dagger}(q,d))[:\hat{n}]$$

$D_q^{-\dagger}$ – negative sample for query q
C – corpus
D_q^{+} – positive documents according to labels
$\hat{n}$ – number of documents to be sampled
(4.3)

JPQ forms a good example of a supervised quantizer. Unfortunately, the model is aimed at retrieval, not re-ranking, so it is not fully applicable to our problem. Having said that, WolfPQ, the model that this thesis proposes, borrows a few ideas from JPQ. The codebook of WolfPQ is initialized with the centroids of PQ to reduce training time. In addition to that, WolfPQ is fine-tuned using end-to-end negative sampling. However, since in the framework of the fast forward indexes retrieval is done using BM25, end-to-end negative sampling in the case of WolfPQ is based on BM25.

4.3. Contextual quantization

As far as we are aware, *Compact Token Representations with Contextual Quantization for Efficient Document Re-ranking* [70] is the only paper that directly touches the topic that this thesis explores, i.e. the development of a quantization algorithm for dense re-ranking. What sparked the authors to investigate quantization for re-ranking was the noticeable success of ColBERT [40], a late interaction dual encoder which managed to bring significant performance improvements. The mathematical details behind ColBERT have already been presented in chapter 2.

As the authors of the paper mention, while ColBERT has brought about significant improvements in terms of both ranking performance and speed, this has come at the cost of extensive memory requirements. As an example, for the MSMARCO corpus [5, 11], the ColBERT encodings take up 1.6TB [70], making it impossible to store it anywhere other than the disk, which results in slow random access. The paper attempts to tackle the issue by proposing *Contextual Quantization (CQ)* [70], a novel quantization technique aimed at compressing the ColBERT dense index for re-ranking.

The core concept that CQ relies upon is something called *contextual decomposition*. The idea behind contextual decomposition is to model the BERT token embeddings as being made up of two components, one that is *context-dependent* and one that is *context-independent*. While the context-dependent components are assumed to be implicitly learned by the model, the context-independent components are explicitly created by feeding single tokens into BERT, and they are shared across multiple documents. The mathematical details behind contextual decomposition can be found in equation 4.4.

$$\begin{aligned}
&E(t_i) = comp(E^{\circ}(t_i), E^{\Delta}(t_i)) \Leftrightarrow (E^{\circ}(t_i), E^{\Delta}(t_i)) = decomp(E(t_i))\\
&E(t_i) = BERT(t_1, t_2, ..., t_n)_i\\
&E^{\circ}(t_i) = BERT(t_i)\\
&\\
&E(t_i) - \text{BERT embedding for term i}\\
&E^{\circ}(t_i) - \text{context-independent BERT embedding for term i}\\
&E^{\Delta}(t_i)) - \text{context dependent BERT embedding}\\
&comp - \text{contextual composition function}\\
&decomp - \text{contextual decomposition function}\\
&d = \{t_1, t_2, ..., t_n\} - \text{document modeled as a bag of terms}
\end{aligned} \tag{4.4}$$

Architecturally, CQ is composed of two functional component, a *quantization encoder* and a *quantization decoder*.

The role of the quantization encoder is to take a term embedding as input and output a discrete index into the associated codebook. More specifically, the quantization encoder takes the concatenation between the standard BERT encoding of a term and its associated context-independent embedding. For each input token, the encoder outputs a vector $s = [s^1, s^2, ..., s^M]$ of M indices into the codebook $C = [C^{(1)}, C^{(2)}, ..., C^{(M)}]$, a tensor of M matrices each encoding the K codewords for one subspace. Since the encoding process needs to be differentiable yet output discrete indices, the encoder relies on *gumbel softmax* [35, 50] as a differentiable sampling function. The architecture of the encoder is presented in figure 4.3.

Symmetrically to the quantization encoder, the quantization decoder involves two steps. The first step consists of using the stored codebook C and indices s to obtain the quantized context-dependent component from codewords. The second step involves combining the quantized context-dependent component and the context-independent component into the final quantized encoding of the term. This process is performed in real-time using a shallow neural network, as shown in figure 4.4.

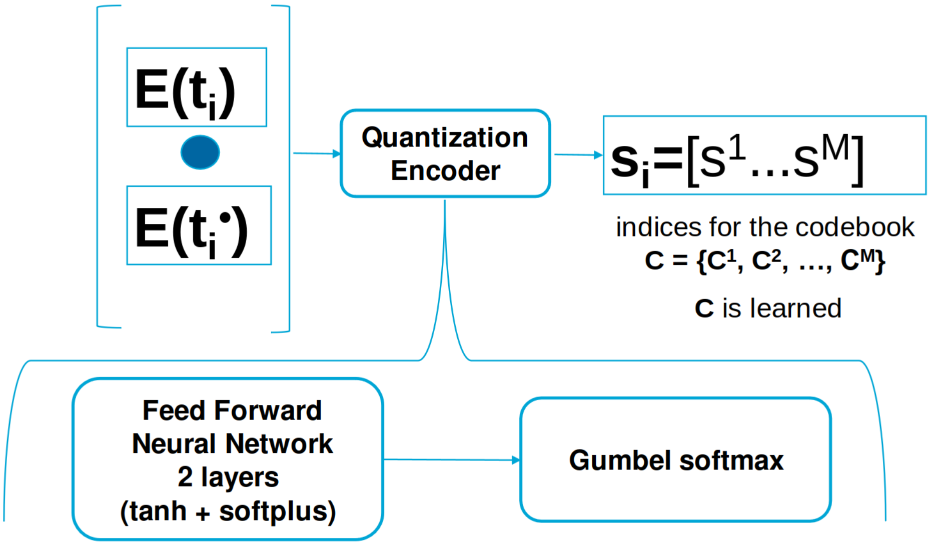


Figure 4.3: Diagram illustrating the architecture of the CQ encoder.

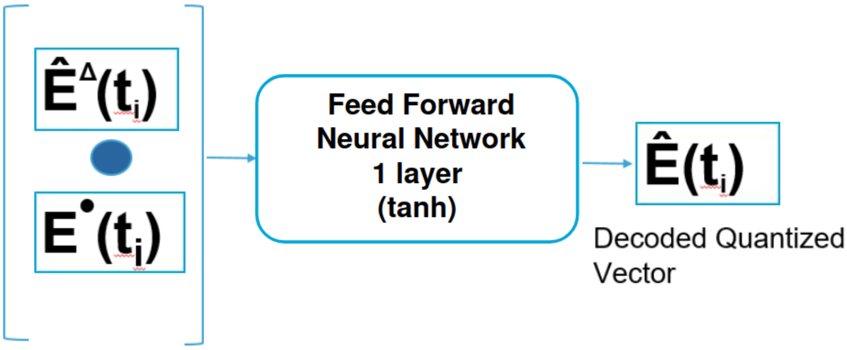


Figure 4.4: Diagram illustrating the architecture of the CQ decoder.

The composition of the quantized context-dependent embedding can be performed either through concatenation, analogously to PQ [36], or thorough addition, in a similar fashion to ADQ [4], as illustrated by equations 4.5 and 4.6.

$$\hat{E}^{\Delta}(t_i) = C^{(1)}_{s^1} \circ C^{(2)}_{s^2} \circ ... \circ C^{(M)}_{s^M} \tag{4.5}$$

$$\hat{E}^{\Delta}(t_i) = C^{(1)}_{s^1} + C^{(2)}_{s^2} + ... + C^{(M)}_{s^M} \tag{4.6}$$

In terms of training, several loss functions have been tried, including mean squared error (mean distortion) and pairwise cross-entropy loss [70, 16]. Despite these, the training procedure that was proven the best is knowledge distillation with a non-quantized version of ColBERT as the teacher model. *Margin-MSE* [32] is the distillation loss that has been used for training, whose formula is shown in equation 4.7.

$$L_{MarginMSE}(q) := \sum_{d^+,d^- \in S(q)}[(f_{qd^+} - f_{qd^-}) - (\hat{f}_{qd^+} - \hat{f}_{qd^-})]^2$$

f_{qd} – dense score for document d computed using ColBERT (4.7)
$\hat{f}_{qd}$ – dense score for document d computed using ColBERT-CQ (quantized ColBERT)
$S(q)$ – pairs of positive and negative samples for query q

CQ represents a good shot at solving the problem of quantization for re-ranking. Its main limitation is the fact that it is specifically designed to work with ColBERT, which is a dual-encoder that produces massive dense indices, to begin with. Due to the fact that **CQ** operates at the token level, there is no obvious way to adapt it to single vector models such as *TCT-ColBERT* or *Aggretriever*. Having said that, WolfPQ, our proposed model, borrows a couple of helpful design ideas from CQ. More specifically, WolfPQ uses the same training procedure for fine-tunning as CQ. On top of that, some architectural elements of the neural network used in semantic sampling are inspired by the CQ encoder component.

5

Vector quantization for re-ranking

The aim of this chapter is to provide the reader with a deep dive into the mathematical details of the techniques used to answer the research questions. To this end, the first section describes the process of applying product quantization for memory-efficient dense re-ranking. This is done to assess the effectiveness of product quantization on single-vector dual encoders, essentially answering the first research question. Building on the methods used in the first research question, the second section looks into leveraging sparse-score interpolation in an attempt to improve the performance of quantized dual-encoder re-rankers. Finally, the last section introduces WolfPQ, a novel data-driven quantization algorithm which aims to achieve higher re-ranking performance while creating an index of the same size as classical product quantization.

5.1. Dense re-ranking with product quantization

As mentioned above, the scope of this section is to provide a deep dive into the mathematical details involved in applying product quantization to single-vector dual-encoder-based re-ranking, which was done to evaluate its effectiveness in accordance with the objectives of the first research question. In our problem setup, we have already provided a general formulation of re-ranking with a quantized dense index, which we reiterate in equation 5.1.

$$\begin{aligned} &\phi : \mathbb{R}^H \to C \\ &rerankingScore(d, q) := \phi(\eta_d(d)) \cdot \eta_q(q) = \hat{\eta_d}(d) \cdot \eta_q(q) \end{aligned} \tag{5.1}$$

In this section, we will take things further and see how we can concretely define the quantizer ϕ and codebook C according to the product quantization (PQ) algorithm [36].

The fundamental idea that lies behind product quantization is dividing the original vector space into lower-dimensional subspaces. This is achieved by breaking up each document encoding vectors into subvectors of equal size, as illustrated by the diagram in figure 5.1, as well as indicated by equation 5.2.

$$\begin{aligned} &\eta_d(d_i) = \eta_d(d_i)^{(1)} \circ \eta_d(d_i)^{(2)} \circ ... \circ \eta_d(d_i)^{(M)} \\ &\eta_d(d_i)^{(j)} = \left[\eta_d(d_i)_{j \cdot \frac{H}{M}}, \eta_d(d_i)_{j \cdot \frac{H}{M}+1}, ..., \eta_d(d_i)_{(j+1) \cdot \frac{H}{M}}\right]^T \\ \\ &\eta_d(d_i)^{(j)} - \text{the jth subvector of the document encoding} \\ &M - \text{the number of subspace divisions} \\ &H - \text{the length of a document encoding vector} \\ &a \circ b - \text{operation representing vector concatenation} \end{aligned} \tag{5.2}$$

After the subspace division has been performed, the next step is to apply quantization in each of the vector subspaces independently. Hence, in product quantization we will define a subquantizer $\phi^{(j)}$

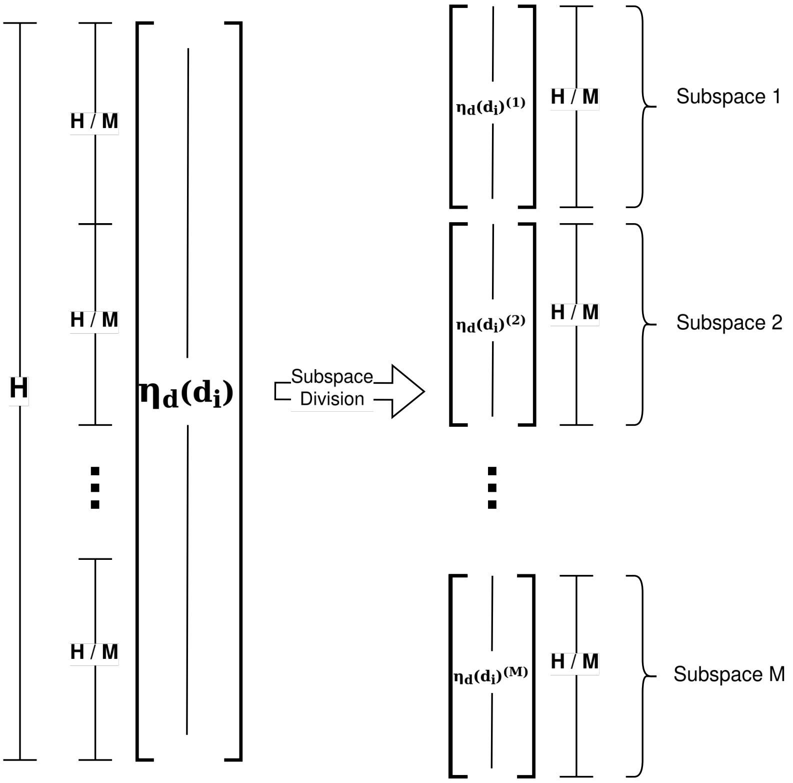


Figure 5.1: Diagram illustrating the division of a vector representing a H-dimensional document encoding ($\eta_d(d_i)$) into M subsvectors ($\{\eta_d(d_i)^{(1)}, \eta_d(d_i)^{(2)}, ..., \eta_d(d_i)^{(M)}\}$) corresponding to the M subspaces used in product quantization.

and subcodebook $C^{(j)}$ for each subspace j. The subcodebook $C^{(j)}$ is obtained by applying k-means clustering on the set of subvectors corresponding to the subspace j. The subcodebook $C^{(j)}$ will be made up of the set of K centroids that result from applying the clustering algorithm. This process is formally described in equation 5.3.

$$C^{(j)} = \{c_1^{(j)}, c_2^{(j)}, ..., c_K^{(j)}\} \leftarrow \texttt{KMeans}(\{\eta_d(d_1)^{(j)}, \{\eta_d(d_2)^{(j)}, ..., \{\eta_d(d_n)^{(j)}\}) \tag{5.3}$$

Having obtained the subcodebook $C^{(j)}$, it is time to come up with a definition for the subquantizer $\phi^{(j)}$. While we could come up with an arbitrary definition for the subquantizer, we need to keep in mind that we aim to perform an optimal quantization of the encodings. In this context, optimality is defined by obtaining a quantized vector as close as possible to the original. In order to formally quantify *closeness* between the original encoding and its quantized form, we will introduce a metric called *distortion*, which consists of the squared euclidean distance between two vectors, as defined by equation 5.4.

$$\begin{aligned} &\texttt{dist}(v_1, v_2) = \|v_1 - v_2\|^2 \\ &\texttt{dist}(v_1, v_2) - \text{the distortion between the vectors } v_1 \text{ and } v_2 \end{aligned} \tag{5.4}$$

Making use of the mathematical formulation of the distortion, we can start defining the objective for our ideal quantization function, which, as mentioned before, should try to minimize the distortion between some arbitrary vector input and its quantized form. This idea leads us to the objective function described by equation 5.5.

$$\begin{aligned} &\phi = \text{argmin}_{\phi^* \in \Phi}\, E_d[\texttt{dist}(\phi^*(\eta_d(d)), \eta_d(d))] = \text{argmin}_{\phi^* \in \Phi}\, E_d[\texttt{dist}(\hat{\eta_d}(d), \eta_d(d))] \\ &\phi - \text{the optimal quantizer} \\ &\Phi - \text{the set of all possible quantizers} \end{aligned} \tag{5.5}$$

If we assume that the documents in our corpus D are a representative sample of all the documents that could be encountered in the wild, equation 5.5 can be approximated by equation 5.6.

$$\begin{aligned}&\phi \approx \operatorname{argmin}_{\phi^* \in \Phi} \sum_i \texttt{dist}(\phi^*(\eta_d(d_i)), \eta_d(d_i)) = \operatorname{argmin}_{\phi^* \in \Phi} \sum_i \texttt{dist}(\hat{\eta_d}(d_i), \eta_d(d_i)) \\ &\phi - \text{the optimal quantizer} \\ &\Phi - \text{the set of all possible quantizers}\end{aligned} \tag{5.6}$$

Equation 5.6 is also known as Lloyd optimality condition [36] and it can be proven that, under the structural constraints imposed by product quantization, its solution is a quantizer that assigns each vector to the nearest centroid in every subspace, as shown by equation 5.7.

$$\begin{aligned}&\phi^{(j)}(\eta_d(d_i)^{(j)}) = \operatorname{argmin}_{c^{(j)} \in C^{(j)}} \texttt{dist}(\eta_d(d_i)^{(j)}, c^{(j)}) \\ &\phi(\eta_d(d_i)) = \phi^{(1)}(\eta_d(d_i)^{(1)}) \circ \phi^{(2)}(\eta_d(d_i)^{(2)}) \circ ... \circ \phi^{(M)}(\eta_d(d_i)^{(M)}) \\ &\phi^{(j)} - \text{the subquantizer for subspace j} \\ &\phi - \text{the full quantizer function}\end{aligned} \tag{5.7}$$

The reason why equation 5.7 describes the optimal quantizer is not hard to see once you realize that the distortion between 2 vectors is equal to the sum of the distortions of the vectors in each of the M subspace divisions, as shown by equation 5.8. Assuming that we took any other codeword rather than the minimum distance one in any of the subspaces, according to equation 5.8, we would obtain a larger distortion than if we took the minimum distance codeword, which would violate Lloyd's optimality condition (equation 5.6). It follows that the quantizer defined in equation 5.7 is indeed optimal in the context of product quantization.

$$\texttt{dist}(v_1, v_2) = \sum_j \texttt{dist}(v_1^{(j)}, v_2^{(j)}) \tag{5.8}$$

Now that we have obtained the optimal quantizer ϕ, we need to define our codebook C. Analyzing equation 5.7, it is not hard to see that the final quantized vector $\hat{\eta_d(d_i)} = \phi(\eta_d(d_i))$ is simply a combination of codewords selected from the M subcodebooks $\{C^{(1)}, C^{(2)}, ..., C^{(M)}\}$. Hence, the final codebook can be expressed as the cartesian product of all the subcodebooks, as described in equation 5.9.

$$C = C^{(1)} \times C^{(2)} \times ... \times C^{(M)} \tag{5.9}$$

5.2. Quantization with sparse interpolation

In the previous section, we saw how product quantization can be applied to our two-stage retrieval setup to reduce the memory used by the re-ranking stage. To this end, we redefined the `rerankingScore` function to take the dot product between the quantized document encoding and the query encoding, as specified by equation 5.1. As we will see in the results section, this leads to a significant drop in re-ranking performance. Ideally, we would like to mitigate the performance drop without introducing any new memory costs to the system. Recent studies suggest that performing linear interpolation of sparse scores with dense scores can lead to better ranking accuracy [68, 44]. Since this operation comes naturally to our setup that already computes BM25 scores for the retrieval step, it made sense to investigate its potential in mitigating quantization performance drops. Performing the linear interpolation between the quantized dense scores and the sparse scores conducts the redefinition of the `rerankingScore` function, as indicated by equation 5.10.

$$\begin{aligned}&rerankingScore(d, q) := (1 - \alpha) \cdot \hat{\eta_d}(d) \cdot \eta_q(q) + \alpha \cdot \texttt{bm25}(d, q) \\ &\alpha \in [0, 1] \\ &\alpha - \text{interpolation parameter}\end{aligned} \tag{5.10}$$

5.3. Supervised quantization for re-ranking

In the previous sections, we have seen how product quantization (PQ) can be applied to a dual-encoder-based dense index to reduce its memory at the expense of some ranking performance decay. We have also seen that the product quantization algorithm minimized the Lloyd optimality condition described in the equation 5.6. While it can be argued that the Lloyd optimality condition is a good objective function for our problem setup, it is debatable whether or not it is the best objective function. This is because, while product quantization optimizes equation 5.6, we measure the performance of a re-ranking system by metrics such as $NDCG$, AP or RR [71].

Considering the above, one can claim that there is an objective gap between product quantization and the re-ranking problem. This leads to the question of whether or not we can mitigate the gap by training a quantizer in a more appropriate way for re-ranking. To test this idea, we propose **WolfPQ**, a differentiable vector quantization algorithm that can be trained in a supervised setting. In the following subsections, we will explore the inner workings of WolfPQ in their full mathematical depth.

5.3.1. General architecture of WolfPQ

As its name suggests, the overall architecture of WolfPQ is inspired by how classical product quantization (PQ) works. Subspace division, one of the core ideas behind product quantization (PQ), is also an operation that WolfPQ relies upon. Hence, the input vector to WolfPQ is divided into subvectors of equal length, as indicated by equation 5.11.

$$
\begin{aligned}
&\eta_d(d_i) = \eta_d(d_i)^{(1)} \circ \eta_d(d_i)^{(2)} \circ \ldots \circ \eta_d(d_i)^{(M)} \\
&\eta_d(d_i)^{(j)} = \left[\eta_d(d_i)_{j \cdot \frac{H}{M}}, \eta_d(d_i)_{j \cdot \frac{H}{M}+1}, \ldots, \eta_d(d_i)_{(j+1) \cdot \frac{H}{M}}\right]^T \\
\\
&\eta_d(d_i)^{(j)} - \text{the jth subvector of the document encoding} \\
&M - \text{the number of subspace divisions} \\
&H - \text{the length of a document encoding vector} \\
&a \circ b - \text{operation representing vector concatenation}
\end{aligned}
\tag{5.11}
$$

What is different from product quantization, however, is how the codebooks are defined. If, in the case of PQ, the codebooks are obtained by applying K-means clustering on the vectors in each of the subspaces, in WolfPQ, codebooks are treated as learnable parameters, similar to the weights of a neural network. This idea is also used by other supervised quantization algorithms, such as JPQ [22]. Hence, one of the parameters that we are learning during the training of WolfPQ is a tensor C that is made up of a stack of $K * \frac{H}{M}$ matrices, representing the codebooks for each of the M subspaces, as illustrated by equation 5.12.

$$
\begin{aligned}
&C = \left[C^{(1)}, C^{(2)}, \ldots, C^{(M)}\right] \\
&C^{(j)} := \left[c_1^{(j)}, c_2^{(j)}, \ldots, c_K^{(j)}\right] \\
\\
&C^{(j)} \in \mathbb{R}^{K \times \frac{H}{M}} - \text{subcodebook for subspace j} \\
&C \in \mathbb{R}^{M \times K \times \frac{H}{M}} - \text{codebook}
\end{aligned}
\tag{5.12}
$$

In addition to the codebooks, WolfPQ introduces another set of learnable parameters representing rotation matrices, one for each subspace. The rotation matrices are inspired by algorithms such as OPQ [26], or JPQ [22], which highlight the effectiveness of rotations in increasing the performance of quantized dense indices. Thus, the second set of learnable parameters in WolfPQ are encoded by the tensor R, containing the learned rotation matrix of each of the subspaces, as shown in equation 5.13.

$$
\begin{aligned}
&R = \left[R^{(1)}, R^{(2)}, \ldots, R^{(M)}\right] \\
&R^{(j)} \in \mathbb{R}^{K \times K} - \text{rotation matrix for subspace j} \\
&R \in \mathbb{R}^{M \times K \times K} - \text{rotation matrices tensor}
\end{aligned}
\tag{5.13}
$$

The quantization process consists of selecting one of the codewords from the rotated subcodebook corresponding to each subspace to encode each corresponding subvector of the input vector. Since we explore multiple ways of performing the selection, we will formally refer to the selection as an abstract function, as introduced by equation 5.14, whose exact definition will be described in the next subsections.

$$\begin{aligned} &Select : \mathbb{R}^{K\times\frac{H}{M}} \times \mathbb{R}^{\frac{H}{M}} \rightarrow \{1, 2, ..., K\} \\ &Select(C^{(j)}, \eta_d(d_i)^{(j)}) := \text{the index of the codeword selected for the subvector } \eta_d(d_i)^{(j)} \end{aligned} \tag{5.14}$$

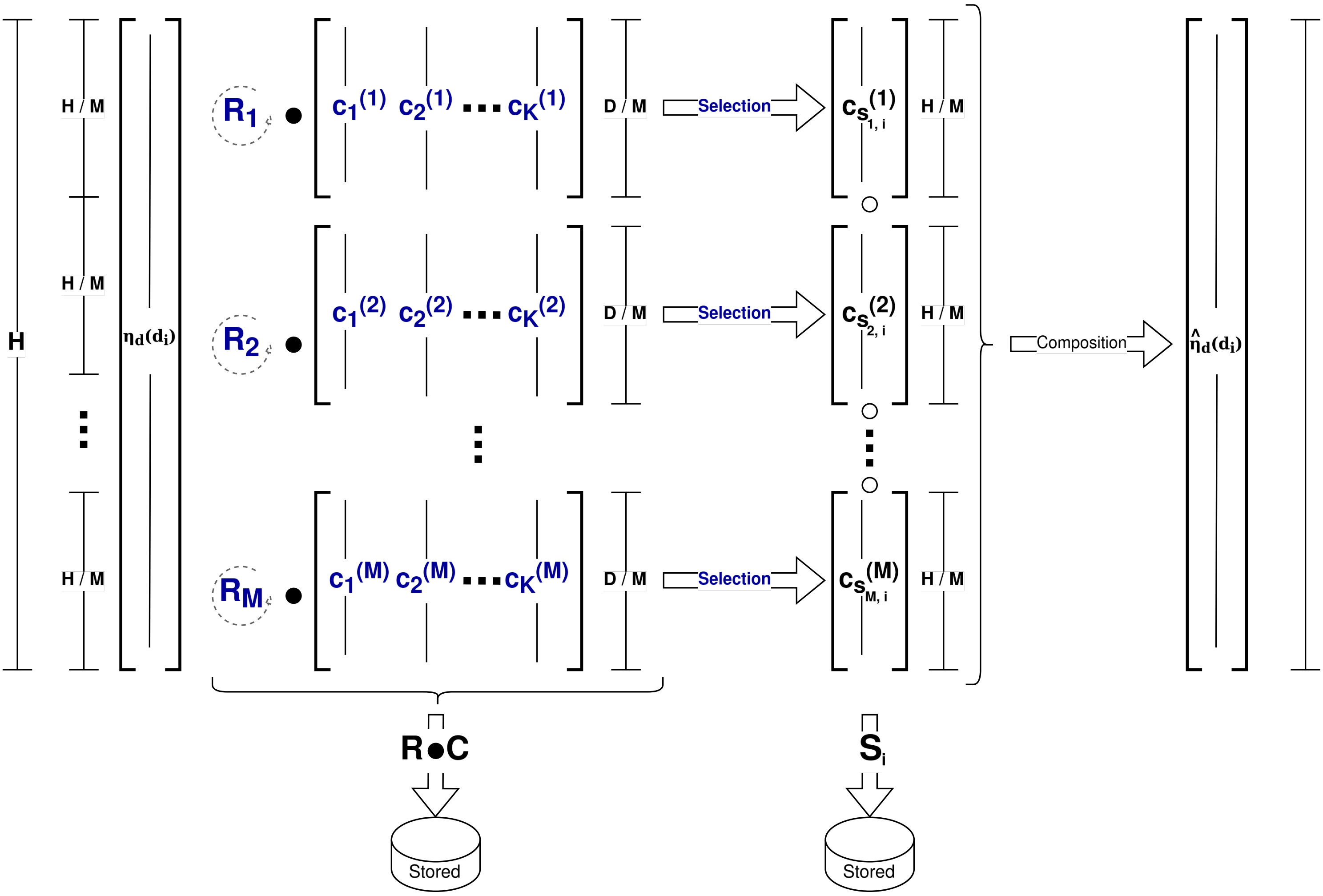


Figure 5.2: Diagram illustrating the general architecture of WolfPQ. The input vector $\eta_d(d_i)^{(j)}$ is divivided into M subvectors of equal size, one subector for each susbspace. In each of the M subspaces, the selection function is applied on the subvector and rotated subcodebook to select a codeword. The selected codewords are than recomposed into the final quantized vector. The rotated subcodebooks and selected indices are stored in memory and they represent the compressed representation of the dense index. The learnable components are highlighted in blue.

Based on the selection function, we can start defining the subquantizer function $\phi^{(j)}$ in the context of WolfPQ. Similar to PQ, the role of the subquantizer function is to return a codeword in each of the M subspaces, which will be later combined to construct the quantized version of the input document embedding $\hat{\eta_d}(d_i)$. Hence, the subquantizer $\phi^{(j)}$ and the selection function (described in equation 5.14) are closely related, the main difference being that the selection function return an index refering to the selected codeword, while the subquantizer returns the selected codeword. One other aspect to keep in mind is that we will not apply the selection function to the original subcodebooks, but rather to their rotated version, obtained by multiplying each rotation matrix to its corrsponding subcodebook. In the light of all of these, we formally define the subquantizer function $\phi^{(j)}$ in equation 5.15.

$$\begin{aligned}
&\phi^{(j)}(\eta_d(d_i)^{(j)}) := R^{(j)} \cdot C^{(j)}[s_{ji}] = R^{(j)} \cdot c^{(j)}_{s_{ji}} \\
&s_{ji} - := Select(R^{(j)} \cdot C^{(j)}, \eta_d(d_i)^{(j)}) \\
&C^{(j)} := \left[c^{(j)}_1, c^{(j)}_2, ..., c^{(j)}_K\right] \\
&\phi^{(j)} - \text{subquantizer function for subspace j} \\
&C^{(j)} - \text{subcodebook for subspace j} \\
&C^{(j)}[l] - \text{indexing operator, extracting the } l\text{th column of the subcodebook} \\
&c^{(j)}_k - \text{kth codeword in the subcodebook for subspace j} \\
&R^{(j)} - \text{rotation matrix for subspace j} \\
&s_{ji} - \text{the index of the selected codeword for document encoding i in susbpace j}
\end{aligned} \tag{5.15}$$

In a similar way to PQ, the final quantized vector $\hat{\eta_d}(d_i)^{(j)}$ is obtained by concatenating the outputs of each of the M subquantizers $\phi^{(1)}, \phi^{(2)}, ..., \phi^{(M)}$, as shown in equation 5.16.

$$\begin{aligned}
&\phi(\eta_d(d_i)) = \phi^{(1)}(\eta_d(d_i)^{(1)}) \circ \phi^{(2)}(\eta_d(d_i)^{(2)}) \circ ... \circ \phi^{(M)}(\eta_d(d_i)^{(M)}) \\
&\phi^{(j)} - \text{the subquantizer for subspace j} \\
&\phi - \text{the full quantizer function}
\end{aligned} \tag{5.16}$$

What is ultimately stored in memory after applying WolfPQ on a dense index is the rotated codebook $R \cdot C$ and the selected codeword indices for each of the quantized document encodings $S_1, S_2, ..., S_n$, as illustrated by figure 5.2. During runtime, the quantized version of the document encodings is reconstructed based on the stored indices and codebook, as already described by the equations 5.15 and 5.16. Since the stored output and quantized vector reconstruction process of WolfPQ follow the same structure as that of PQ, WolfPQ can be considered functionally equivalent to PQ once it has been trained. This is because WolfPQ will always create a compressed index of the same size as PQ, which can be used in the same manner by the integrated system, making PQ and WolfPQ fully interchangeable. The main difference between PQ and WolfPQ is that WolfPQ will be trained to create a compressed, dense index better adapted for the re-ranking task.

5.3.2. Selection by minimum distance

In the previous section, we presented the overall architecture and flow of WolfPQ, the supervised quantization algorithm proposed for our re-ranking problem. As the reader might have noticed, the subsection introduced the subquantizer $\phi^{(j)}$ as a function that depends on an abstract $Select$ operation that has not been defined. In this subsection and the next, we will remedy this by presenting two alternative ways in which the selection operation can be defined.

The simplest way to define the selection operation is to simply follow the recipe used by product quantization [36], as well as other well-known vector quantization algorithms [26, 53], and always select the codeword that is the closest to the subvector in each of the subspaces. Since this selection method is well-tested and is known to usually provide decent results, it makes sense to choose it as our baseline method. The formal definition of *selection by minimum distance* is provided by equation 5.17.

$$Select_{MinDist}(R^{(j)} \cdot C^{(j)}, \eta_d(d_i)^{(j)}) := \text{argmin}_{s \in \{1,2,...,K\}} \texttt{dist}(R^{(j)} \cdot c^{(j)}_s, \eta_d(d_i)^{(j)}) \tag{5.17}$$

Having defined our selection operation, the construction of the output quantized vector $\eta_d\hat{(d_i)}$ can be implemented in a differentiable way by building sparse, one-hot-encoded vectors that specify the selected indices. The target codeword for each of the M subspaces is retrieved by multiplying the codeword's subcodebook with the corresponding sparse selection vector. The result of the multiplication will be a matrix of the same shape as the subcodebook, filled in with 0s in all positions that do not correspond to the target, selected codeword. Summing all the columns of this sparse output matrix will, hence, yield exactly the selected codeword, but after a sequence of differentiable operations. All of these lead us to the differentiable redefinition of our subquantizer function, as expressed by equation 5.18.

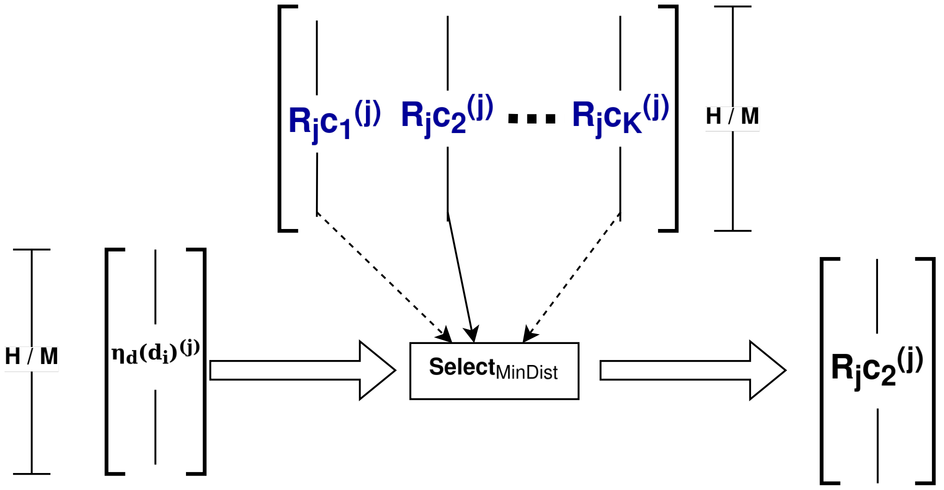


Figure 5.3: Diagram illustrating the selection by minimum distance procedure. The distance between the subvector $\eta_d(d_i)^{(j)}$ and all of rotated codewords $\{R_jc_1^{(j)}, R_jc_2^{(j)}, ..., R_jc_K^{(j)}\}$ is computed and the rotated codeword with the minimun distance is selected (in the example, the second codeword has the minimum distance).

$$\begin{aligned} &\phi^{(j)}(\eta_d(d_i)^{(j)}) := R^{(j)}C^{(j)} \cdot \texttt{OneHot}(s_{ji}) = \\ &= R^{(j)}\left[c_1^{(j)}, ..., c_{s_{ji}}^{(j)}, ..., c_K^{(j)}\right] \cdot \left[0, ..., 1, ..., 0\right] = R^{(j)}c_{s_{ji}}^{(j)} \\ &s_{ji} - := Select_{MinDist}(R^{(j)}C^{(j)}, \eta_d(d_i)^{(j)}) \end{aligned} \tag{5.18}$$

While the differentiable redefinition of the subquantizer as expressed by equation 5.18 is computationally more expensive than the original definition of equation 5.15, it is needed during the training process of WolfPQ. The reason for this is that differentiability is required for backpropagation, which is the training algorithm employed by WolfPQ and most modern deep-learning models.

5.3.3. Selection by semantic sampling

In the previous subsection, we presented WolfPQ's baseline selection method. This section will explain a more advanced selection procedure, dubbed *selection by semantic sampling*. The intuition driving this selection method is based on the observation that dense re-ranking is ultimately based on analyzing the semantic information contained in the document and the query. Following up on this idea, if we want to train a quantizer that performs better on the re-ranking task, exposing the quantizer to semantic information about the document might be helpful. For instance, if two documents belong to the same topic and are likely to end up in the same retrieval set, their quantized encodings should probably be distinct so that the re-ranker can distinguish between them.

In light of the ideas presented above, we can identify a key limitation of *selection by minimum distance*, i.e. the fact that the selection of the codebook is based purely on vector geometry. While one could argue that in the semantic vector space created by the document encoder η_d, vector geometry corresponds to semantic relationships, we need to keep in mind that WolfPQ, similarly to other vector quantization algorithms [53], relies on subspace division. This means that the inputted semantic vector is broken down into subvectors, which might no longer have well-defined semantic meanings.

To mitigate the limitations presented in the previous paragraph, it becomes necessary to take into account the whole semantic vector when selecting the codeword in each of the subspaces. This leads to our proposed technique, called *selection by semantic sampling*. Roughly inspired from the architecture of CQ's encoder component [70], the semantic sampling selection function relies on a 2-layer feed-forward neural network to analyze and derive useful information from the semantic encoding $\eta_d(d_i)$, which we are aiming to quantize.

The first layer of the neural network has an input size of H, corresponding to the dimensions of the document encodings, and an output size of $M * K/2$, where M is the number of subspaces and K the number of codewords per subspace. The layer uses a $\tanh$ activation function, as described by equation 5.19.

$$\begin{aligned} & h = \tanh(W_1 \cdot \eta_d(d_i) + b_1) \\ & h \in \mathbb{R}^{\frac{MK}{2}} - \text{hidden layer} \\ & \eta_d(d_i) \in \mathbb{R}^H - \text{input semantic vector} \\ & W_1, b_1 - \text{learnable weights and bias} \end{aligned} \tag{5.19}$$

The second layer of the feed-forward network has an input size of $M * K/2$ and an output size of $M * K$. The choice of the output size is not a coincidence, however, since the output logits of the network are interpreted as being proportional to the probability of selecting each of the K codewords in each of the M subspaces. Unlike the first layer of the network, the second layer will use a $relu$ activation function [20], as indicated by equation 5.20. One small aspect to notice is that the original implementation of CQ's encoder used a $softplus$ activation. The reason why we decided to replace $softplus$ with $relu$ is that deep learning literature considers it a better alternative [27] [28, Chapter 6].

$$\begin{aligned} & o = \text{relu}(W_2 \cdot h + b_2) \\ & h \in \mathbb{R}^{\frac{MK}{2}} - \text{hidden layer} \\ & o \in \mathbb{R}^{MK} - \text{output layer} \\ & W_2, b_2 - \text{learnable weights and bias} \end{aligned} \tag{5.20}$$

As mentioned earlier, we interpret the output of the feed forward network as indicating the probabilities of selecting each of the codewords in each of the subspaces. As a consequence of this, we will reshape the output vector $o \in \mathbb{R}^{MK}$ as a 2-dimensional matrix $A \in \mathbb{R}^{M \times K}$, where each row corresponds to the logits encoding the selection probability within one subspace, as shown by equation 5.21.

$$\begin{aligned} & A := \begin{bmatrix} o_1 & o_2 & \cdots & o_K \\ o_{K+1} & o_{K+2} & \cdots & o_{2K} \\ \vdots & \vdots & \dots & \vdots \\ o_{(M-1)K} & o_{(M-1)K+1} & \cdots & o_{MK} \end{bmatrix} \\ & A^{(j)} = \begin{bmatrix} o_{(j-1)K} & o_{(j-1)K+1} & \cdots & o_{jK} \end{bmatrix} \\ & A - \text{reshaped output layer encoding the semantic selection probability logits for all M subspaces} \\ & A^{(j)} - \text{vector encoding the semantic selection probability logits for the jth subspace} \end{aligned} \tag{5.21}$$

While in an ideal scenario, we would only rely on the semantic probability logits derived in equation 5.21, learning a good selection strategy completely from scratch could be a difficult process that would require many optimization steps and a large training dataset. However, since we already know that, while maybe not optimal, selection by minimum distance is a decent strategy, we can design our deep learning architecture in such a way that our model is implicitly biased to select the minimum distance codeword with a higher probability. In the case of WolfPQ, this is implemented by performing a linear interpolation between the semantic selection probability logits $A^{(j)}$ and the one hot encoded minimum distance index $\texttt{OneHot}(Select_{MinDist}(R^{(j)} \cdot C^{(j)}, \eta_d(d_i)^{(j)}))$, as derived in the previous subsection (equation 5.17). The one-hot encoded minimum distance index is multiplied by the minimum distance factor MDF, a scalar hyperparameter that controls the importance given to the minimum distance in the semantic sampling procedure. All of these lead to equation 5.22, which describes the formula for the final semantic probability logits used to sample codewords from the subcodebooks.

$$L^{(j)} := (1 - \gamma) \cdot A^{(j)} + \gamma \cdot MDF \cdot \texttt{OneHot}(Select_{MinDist}(R^{(j)} \cdot C^{(j)}, \eta_d(d_i)^{(j)}))$$

$L^{(j)}$ – final semantic probability logits for codeword sampling
γ – interpolation parameter between pure selection by semantic sampling and selection by minimum distance
MDF – hyperparameter controlling the reliance on selection by minimum distance

(5.22)

While MDF is a hyperparameter, which means that it is chosen by the user before the training process even begins, γ is learned during training. However, since γ is used to perform linear interpolation, it is important that its values remain constrained in the interval $[0, 1]$. To guarantee this, γ is modelled as the output of a logistic function applied on a learned parameter ξ, as shown in equation 5.23. Hence, the equation for the final semantic probability logits becomes equation 5.24, with ξ being the actual parameter learned during training.

$$\gamma = \frac{1}{1+e^{-\xi}}$$

ξ – trainable parameter

(5.23)

$$L^{(j)} := (1 - \frac{1}{1+e^{-\xi}}) \cdot A^{(j)} + \frac{1}{1+e^{-\xi}} \cdot MDF \cdot \texttt{OneHot}(Select_{MinDist}(R^{(j)} \cdot C^{(j)}, \eta_d(d_i)^{(j)}))$$

$L^{(j)}$ – final semantic probability logits for codeword sampling in subspace j (5.24)
ξ – trainable parameter
MDF – hyperparameter controlling the reliance on selection by minimum distance

Having obtained the final semantic probability logits for each of the subspaces, the last step of WolfPQ is to use the logits to sample a codeword from each of the subcodebooks associated with each of the subspaces. In other words, we need to turn each vector of logits $L^{(j)}$ into an index s_{ji} for each subcodebook $C^{(j)}$. Since the feedforward neural network and the parameter ξ are trainable, we need to perform this operation in a differentiable way. One technique that can be applied to perform differentiable sampling from a categorical probability distribution is gumbel-softmax [35, 50]. Since gumbel-softmax has also been employed by CQ [70] to add differentiability to their algorithm, it also became our design choice for WolfPQ, as indicated by equation 5.25.

$$s_{ji} = \text{argmax}_{1 \leq k \leq K} \frac{\exp(\log(L_k^{(j)} + \epsilon_k)/\tau)}{\sum_{k'=1}^{K} \exp(\log(L_{k'}^{(j)} + \epsilon_{k'})/\tau)}$$
$$\epsilon_k \sim \texttt{Gumbel}(0, 1)$$

ϵ_k – noise sampled from a gumbel distribution
τ – temperature parameter
s_{ji} – sampled codeword index

(5.25)

While gumbel-softmax is useful during training due to its property of making sampling differentiable, it also adds stochasticity to the final result, which is not ideal for reproducibility and stability during inference. As a result of this, the method for obtaining the codebook indices s_{ji} described in equation 5.25 is only used during training. During inference, WolfPQ simply takes the index of the maximum logit, as described by equation 5.26, which is not a differentiable procedure, but it is deterministic.

$$s_{ji} = \text{argmax}_{1 \leq k \leq K} L_k^{(j)} \qquad (5.26)$$

As in the case of selection by minimum distance, the index obtain in either equation 5.25 or equation 5.26 is used to select a codeword from the subcodebook of the current subspace as part of the definition of the subcodebook $\phi_{(j)}$ for the subspace j, as indicated by equation 5.27.

$$\begin{aligned}\phi^{(j)}(\eta_d(d_i)^{(j)}) &:= R^{(j)} \cdot C^{(j)} \cdot \texttt{OneHot}(s_{ji}) = \\ &= R^{(j)} \cdot \left[c_1^{(j)}, ..., c_{s_{ji}}^{(j)}, ..., c_K^{(j)}\right] \cdot \left[0, ..., 1, ..., 0\right] = R^{(j)} \cdot c_{s_{ji}}^{(j)}\end{aligned} \tag{5.27}$$

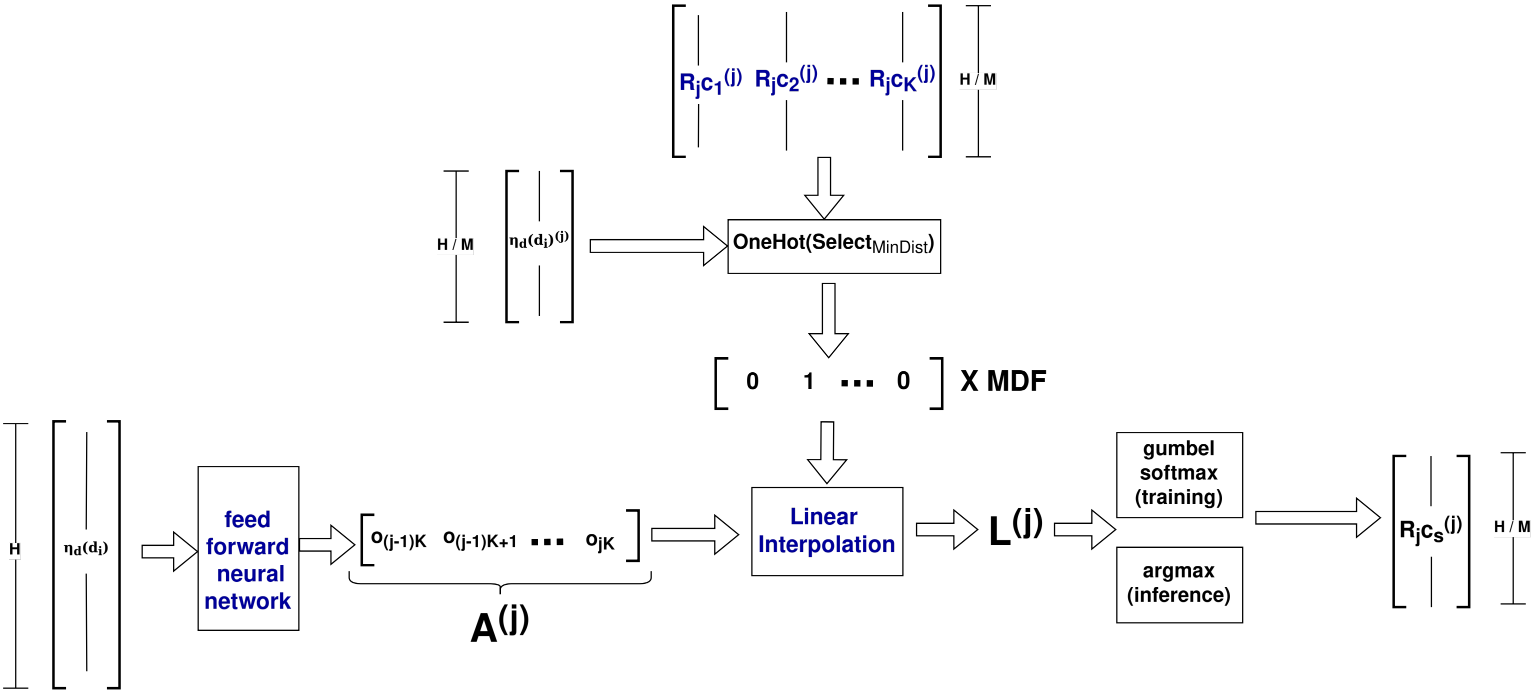

$R_jc_1^{(j)}$ $R_jc_2^{(j)}$ ... $R_jc_K^{(j)}$
H / M
H / M
$\eta_d(d_i)^{(j)}$
OneHot(Select$_{MinDist}$)
[0 1 ... 0] X MDF
H
$\eta_d(d_i)$
feed forward neural network
$o_{(j-1)K}$ $o_{(j-1)K+1}$ ... o_{jK}
$A^{(j)}$
Linear Interpolation
$L^{(j)}$
gumbel softmax (training)
argmax (inference)
$R_jc_s^{(j)}$
H / M

Figure 5.4: Diagram illustrating selection by semantic sampling. The output of minimum distance selection is linearly interpolated with the output of the neural network extracting purely semantic information, to obtain $L^{(j)}$, a vector of logits proportional to the sampling probability for the subcodebook associated to subspace j. The codeword is sampled from the probability distribution described by $L^{(j)}$ using either the differentiable *gumbel-softmax*, during training, or the deterministic *argmax*, during inference.

5.3.4. Pre-training procedure

The previous subsections have provided a deep theoretical overview of the architecture of our proposed supervised quantizer, WolfPQ. Building on this knowledge, the current subsection and the next will tackle technical details regarding the process of training WolfPQ. Not unlike many other deep-learning models, WolfPQ is trained in two stages, *pre-training* and *fine-tuning*. Thus, this section will give a description of the pre-training procedure, while the next will move on to fine-tuning.

The entire design process of WolfPQ followed the idea that in order to obtain a quantizer that outperforms PQ, it is essential to start with a differentiable quantizer with a similar base performance as PQ and then further train it in a supervised fashion to make it *adapt* to the re-ranking task. Following this idea, the aim of pre-training is to obtain an instance of WolfPQ with a similar base performance as PQ while the adaptation to the re-ranking task is performed during fine-tuning.

Since the goal is to obtain a version of WolfPQ that works similarly to PQ, WolfPQ will be pre-trained with the same goal as PQ, i.e. minimizing the distortion between the original vector and its quantized version, as shown in equation 5.28 indicating the loss function used during pre-training. One interesting aspect to notice is that since WolfPQ aims to reconstruct its original input, WolfPQ can be seen as an auto-encoder [12] in its pre-training phase.

$$L(D) := \tfrac{1}{|D|} \textstyle\sum_{d \in D} \|\eta_d(d) - \texttt{WolfPQ}(\eta_d(d))\|^2 \tag{5.28}$$

One other aspect that is important to mention is parameter initialisation, more specifically, the initialisation of the learnable subcodebooks and associated rotation matrices. While random initialisation would be possible, empirical experiments lead to the conclusion that using the subcodebooks of a trained PQ instance and initialising the rotation matrices with the identity matrix, leads to faster convergence than random initialisation. The empirical results make sense intuitively, since we already know that PQ is a decent algorithm for the task.

5.3.5. Fine-tuning procedure

Having seen an overview of pre-training in the previous subsection, we will now move on to fine-tuning. As previously mentioned, the role of fine-tuning is to make WolfPQ adapt to the re-ranking task in the hope of obtaining better performance. Hence, the core idea behind fine-tuning is to perform the training of the quantizer in an end-to-end fashion using a goal that is better aligned to re-ranking than mean distortion (equation 5.28). In addition to this, having seen that sparse interpolation benefits quantization, a secondary aim is to increase how much WolfPQ gains from sparse interpolation compared to classical PQ. To accommodate both of these objectives, instances of WolfPQ were trained on end-to-end settings, both with and without interpolation, and the results were compared.

The loss function chosen to train WolfPQ was borrowed from the paper introducing CQ [70]. Experiments on CQ have shown that the training method which leads to the best results involves knowledge distillation. More specifically, a non-quantized version of the dual-encoder is used as the teacher model, while the student model is the same dual-encoder combined with WolfPQ. This leads to the MarginMSE loss function [32] in equation 5.29.

$$
\begin{aligned}
&L_{MarginMSE}(q) := \textstyle\sum_{d^+,d^- \in S(q)} [(f_{qd^+} - f_{qd^-}) - (\hat{f}_{qd^+} - \hat{f}_{qd^-})]^2 \\
&f_{qd^+} := \eta_q(q) \cdot \eta_d(d^+) \\
&f_{qd^-} := \eta_q(q) \cdot \eta_d(d^-) \\
&\hat{f}_{qd^+} := \eta_q(q) \cdot \texttt{WolfPQ}(\eta_d(d^+)) \\
&\hat{f}_{qd^-} := \eta_q(q) \cdot \texttt{WolfPQ}(\eta_d(d^-)) \\
&S(q) - \text{pairs of positive and negative samples for query } q
\end{aligned}
\tag{5.29}
$$

One problem that one might notice with the loss function in equation 5.29 is that it is not influenced at all by the introduction of sparse interpolation. This is due to the fact that for both the quantized and non-quantized instances of the dual-encoder, the sparse scores remain the same, which would make them cancel out during the computations. Thus, if our aim is to make WolfPQ benefit more from sparse interpolation than PQ, we need to train it using a loss function which is sensitive to sparse interpolation. Since sparse interpolation ultimately influences the final ranking of the documents, one such loss would be a listwise loss, which compares the rank outputted by the model with the expected, ideal rank. While many types of listwise losses exist, we chose to use RankNet [8] due to its popularity.

From a theoretical perspective, good arguments can be made for both the MarginMSE loss function [32], due to its good performance on CQ [70], as well as for RankNet [8], due to its potential to make WolfPQ more adapted to sparse score interpolation. Hence, we have decided to train WolfPQ by merging both losses via linear interpolation, as shown in equation 5.30.

$$
\begin{aligned}
&L(q, rank) := (1 - lip) \cdot L_{MarginMSE}(q) + lip \cdot L_{RankNet}(q, rank) \\
&lip \in [0, 1] - \text{loss interpolation parameter}
\end{aligned}
\tag{5.30}
$$

A final aspect that needs to be discussed is negative sampling, or more specifically, how are the document pairs (d^+, d^-) used during training for the computation of the loss function obtained. It turns out that, according to information retrieval literature, negative sampling is still an area under research, with many alternatives possible [69, 60]. Since negative sampling is out of the scope of our research, we decided to go for the simple form of sampling, which involves taking the lowest-ranked documents in the top 100 BM25 results. Hence, it is very much possible that better sampling techniques could be used, and what is the best sampling technique for training quantizers is a research question that is still unanswered.

6
Experimental setup

The role of this chapter is to provide the reader with an overview of all the relevant details behind the experiments that were performed to pursue the research questions. To this end, the first section will review the metrics used to evaluate passage ranking. Following that, the second section will present the datasets used for experimentation. In contrast, the third section will give some core implementation details to guarantee the reproducibility of results. Finally, the last section will expose the reasoning behind choosing the experimental baselines.

6.1. Metrics for evaluation

As passage ranking is a well-studied and well-defined problem in the field of information retrieval, there are a couple of evaluation metrics that have become a field standard [71]. Naturally, our research adopted the same metrics in order to guarantee that our results are comparable. This section will provide the mathematical details behind the chosen evaluation metrics while explaining why these metrics are suitable.

As already mentioned in the previous chapters of the thesis, the goal of passage ranking is to order passages with respect to their relevancy to a query. This means that a passage ranker performs well if it generally ranks relevant documents higher than non-relevant ones. To be able to quantify that, it is important that we formally define relevancy. It is to this end that we introduce the $rel(q, d)$ function in equation 6.1, which measures relevancy either on a binary (*relevant*, *non-relevant*) or, more rarely, ternary (*highly relevant*, *relevant*, *non-relevant*) scale, depending on the specific dataset.

$$rel(q,d) = \begin{cases} 1 & \text{document } d \text{ is relevant for query } q \\ 0 & \text{document } d \text{ is not relevant for query } q \end{cases} \tag{6.1}$$

The $rel(q, d)$ function forms the basis for the definition of many evaluation metrics, a common one being precision, as defined in equation 6.2.

$$\begin{gathered} Precision(R,q) = \frac{\sum_{(i,d)\in R} rel(q,d)}{|R|} \\ R - \text{ranked list of documents} \\ \text{i.e. list of tuples } (i,d) \text{ where } i \text{ is the rank(position) and } d \text{ is the document} \end{gathered} \tag{6.2}$$

It is prevalent that the precision is evaluated at a cut-off point k, which means that only the first k elements of the ranked list are considered, as shown in equation 6.3. One considerable disadvantage of the precision metric is that it does not take into account the rank positions beyond the cut-off point k. This led to the definition of the average precision, as indicated in equation 6.4. As the reader will see in the results section, average precision was one of the metrics chosen for evaluation.

$$Precision@k(R,q) = \frac{\sum_{(i,d)\in R, i\leq k} rel(q,d)}{k} \tag{6.3}$$

$$AP@k(R,q) = \frac{\sum_{(i,d)\in R, i\leq k} Precision@i(R,q)\cdot rel(q,d)}{\sum_{d\in D} rel(q,d)} \tag{6.4}$$

Another simple yet common evaluation metric that we used for our experiments was reciprocal rank. The reciprocal rank is computed by taking the inverse of the rank corresponding to the first relevant document, as shown in equation 6.5.

$$\begin{aligned} RR(R,q) &= \frac{1}{rank(R,q)} \\ rank(R,q) &= \min\{i|(i,d)\in R \wedge rel(q,d) > 0\} \end{aligned} \tag{6.5}$$

A well-known evaluation metric we adopted is *normalized discounted cumulative gain ($nDCG$)*. Unlike the other metrics, $nDCG$ was explicitly created to fit relevance functions with several grades. A quantity needed to define $nDCG$ is the *discounted cumulative gain (DCG)*, which is a measure of the value the user gets from a particular ranking result, as indicated by equation 6.6.

$$DCG@k(R,q) = \sum_{(i,d)\in R, i\leq k} \frac{2^{rel(q,d)}-1}{\log_2(i+1)} \tag{6.6}$$

As the name suggests, $nDCG$ is obtained from DCG by introducing normalization. This translates into dividing DCG by an ideal discounted cumulative gain $IDCG$ (equation 6.7), computed on a ranked list that is rearranged to always begin with the most relevant documents.

$$nDCG@k(R,q) = \frac{DCG@k(R,q)}{IDCG@k(R,q)} \tag{6.7}$$

6.2. Dataset

The experiments of this research were performed on the MSMARCO passage dataset [5, 11]. This is a very well-known dataset in the information retrieval community that is representative of web search. The dataset contains the following:

- a corpus of roughly 8.8 million passages extracted from web documents.
- sets of search queries or questions
- relevance judgements indicating which passages are relevant for which queries

The MSMARCO passage dataset is complemented by the TREC DL 19 [17] and TREC DL 20 [18], which contain sets of queries that can be used as training and testing datasets for deep learning models.

Given the considerable size of the MSMARCO passage dataset, it can be assumed that an information retrieval that performs well on it will also do well as a real-life search engine.

6.3. Implementation details

The implementation of the experiments relied on the libraries and dependencies listed below.

- **Faiss** [21] - Faiss is an open-source vector similarity search library developed by Meta's research team. The library contains, among other things, implementations of many popular vector quantization algorithms. Our experiments relied on faiss's implementation of PQ.
- **Lucene search** [48] - Lucene search is a high-performance open-source search engine written in Java but having Python wrappers. The experiments made use of Lucene search's implementation of BM25.

- **Hugging face transformers** [33] - Hugging face transformers is a widely popular library for natural language processing. It contains architectures and trained instances of many neural language models. The experimental setup relied on hugging face transformers as a provider for our target dual encoders, i.e. TCT-ColBERT and Aggretriever.
- **HDF5** [31] - HDF5 is an open-source file format for storing large amounts of heterogeneous data. Since the encoded MSMARCO Passage corpus ended up being 27.5GB, the use of HDF5 for efficient storage and random access of semantic vectors became necessary.
- **PyTorch** [59] - WolfPQ was implemented in PyTorch, one of the most popular deep-learning frameworks.
- **ir-measures** [34] - Ir-measures was the Python library that was used for the evaluation metrics of the experiments, as it contains implementations for many metrics commonly used in information retrieval.

The codebase used for the evaluation of the quantization algorithms relied to a large extent on the fast-forward-indexes library [23, 44], which was extended to support quantized indices of various types. The quantization of the dense indices was done on Delft Blue [19], TU Delft's SLURM computer cluster, due to its high parallelism capabilities. The training of WolfPQ was achieved with the help of the premium version of Google Colab [29], which gave us access to fast, high-memory GPUs, specifically Tesla T4 and L4. The full codebase used in this research can be found on the associated open source GitHub repository[1].

6.4. Data pre-processing

The time-efficient fine-tuning of WolfPQ required a lot of data pre-processing, the steps of which are summarized below.

1. *The queries without any associated relevant documents were filtered out of the training set.*
2. *For each query in the remaining training set, the top 100 most relevant documents, according to BM25, were fetched.*
3. *Out of each set of 100 retrieved documents, the least 32 relevant documents were selected as negative samples.*
4. *For each training query, 32 pairs of positive-negative (d^+, d^-) document samples with their associated BM25 scores were formed. For queries with only one relevant document, all 32 irrelevant documents were used in pair-forming. For queries with more than one positive document, the pair were formed by uniformly sampling the irrelevant documents.*
5. *The full 27.5GB TCT-ColBERT dense index encoding all the MSMARCO passages was loaded in the memory of a high-performance Google Colab machine to facilitate fast access.*
6. *For each training query, tensors containing the vector encodings of the formed positive-negative document samples were constructed.*
7. *The pre-processed dataset containing the queries (with their associated encodings), positive-negative pairs of document ids, their associated vector encodings, as well as the BM25 scores was saved on the hard-disk in chunks of roughly 1.8GB each.*
8. *A data structure mapping query ids to chunks was constructed and saved on the hard disk together with the chunks.*
9. *WolfPQ was trained with the help of a simple caching system that kept the most recently used chunk in memory until a new chunk needed to be accessed.*

6.5. Baselines

The final results were obtained by performing re-ranking with the help of PQ-quantized and WolfPQ-quantized dense indices, which were compared against the baselines specified below

[1] `https://github.com/catalinlup/Quantization-for-compact-neural-passage-re-ranking`

- **BM25** - widely used sparse retrieval method [61]. Having BM25 as part of the baselines is crucial since any dense re-ranking setup that is outperformed by BM25 is useless, as the purpose of re-ranking is defeated.
- **BM25 + TCT-ColBERT and BM25 + Aggretriever** - original, non-quantized, single-vector dual-encoder re-ranking setups [61, 46, 47]. They induce the upper bound for our results and enable us to measure the relative performance drop caused by quantization.
- **BM25 + ColBERT-PQ** - the baseline was taken from the contextual quantization paper [70]. Its relevancy comes from the fact that it enables us to compare the effectiveness of PQ on single-vector dual-encoders (our experiments) against its effectiveness on ColBERT.
- **BM25 + ColBERT-CQ** - the model proposed by the contextual quantization paper [70]. This baseline is important because it represents the only attempt of developing a supervised quantizer for re-ranking. Hence, it is the only model that is directly comparable to WolfPQ.
- **TCT-ColBERT-PQ** - dense retrival using a PQ-quantized [36] TCT-ColBERT [47] index. Having it as a baseline enables us to highlight the importance of the sparse retrieval component of our setup.
- **JPQ and RepCONC** - state-of-the-art quantization techniques for dense retrieval [22, 72]. They are useful since they enable us to compare our re-ranking-oriented setup with a quantized dense retrieval setup.

7 Results

This chapter will provide the reader with a presentation of the experimental results of the research, as well as a discussion of how these results relate to and answer our research questions. To this end, this chapter is structured in a way that mirrors our research questions, objectives and contributions. Hence, the first section describes the results obtained from the application of product quantization (PQ) on single-vector dual-encoders to answer the first research question. The following section dives into the results obtained while investigating the second research question related to the intersectionality between quantization and sparse-score interpolation. Last but not least, the third section presents the results obtained from the evaluation of WolfPQ, our proposed novel quantization algorithm, establishing its performance and capabilities with respect to relevant baselines.

7.1. Unsupervised quantization

- **RQ1**: How well does product quantization work with single-vector dual encoders, specifically TCT-ColBERT and Aggretriever?

To answer the first research question, we applied product quantization (PQ) [36] to the dense indices obtained by encoding the MSMARCO Passage corpus [5, 11] with the TCT-ColBERT [47] and the Aggretriever [46] dual-encoders. The original index size for both dual-encoders was *27.2GB*. Multiple compressed indices were obtained by applying PQ with different numbers of subspace divisions M and codebook sizes K. The passage re-ranking performance of the setups built on top of the various compressed indices was evaluated in an end-to-end fashion by looking at the *RR@10*, *AP@1000* and *NDCG@10* metrics on the MSMARCO PASSAGE DEV [5, 11], TREC DL 2019 [17] and TREC DL 2020 [18]datasets.

7.1.1. Hyperparameter exploration

As specified above, various initializations of the number of subspaces M and codebook sizes K were tried to assess the impact of PQ at various levels of compression. More specifically, we varied K from 256 to 4096 and M from 8 to 96. To obtain an optimum bit-level representation of the codebook indices, it was necessary that the chosen values of K were a power of 2. Similarly, since the size of the vectors outputted by both dual-encoders is 768, the values of M needed to be divisors of 768. Both M and K influence the size of the quantized index, with higher values corresponding to larger, less compressed indices, as indicated by figure 7.1.

As one would normally expect, higher levels of compression of the dense index correspond to lower re-ranking scores. As the degree of compression decreases, the re-ranking performance increases, converging asymptotically to the performance of the original, non-compressed index. This behaviour is shown in figure 7.2, which illustrates the re-ranking performance of PQ-quantized dense indices with respect to the index size.

One interesting aspect to notice is that the performance of the Aggretriever index seems to be impacted less by compression, as shown by both figure 7.2 and table A.1. For instance, while the $RR@10$ score measured on the dev set of the Aggretriever is only roughly 2.5% higher than the one of TCT-ColBERT (0.368 vs. 0.359), the differences between Aggretriever-PQ and TCT-ColBERT-PQ at a compression factor of x163.85 is 70%. As the degree of compression decreases, so does the difference between the Aggretriever-PQ and TCT-ColBERT-PQ, but this phenomenon can be observed at all degrees of compression. Hence, we can conclude that the Aggretriever is a more *quantizable* than TCT-ColBERT, at least when it comes to applying PQ. The reasons for this need to be investigated by future research.

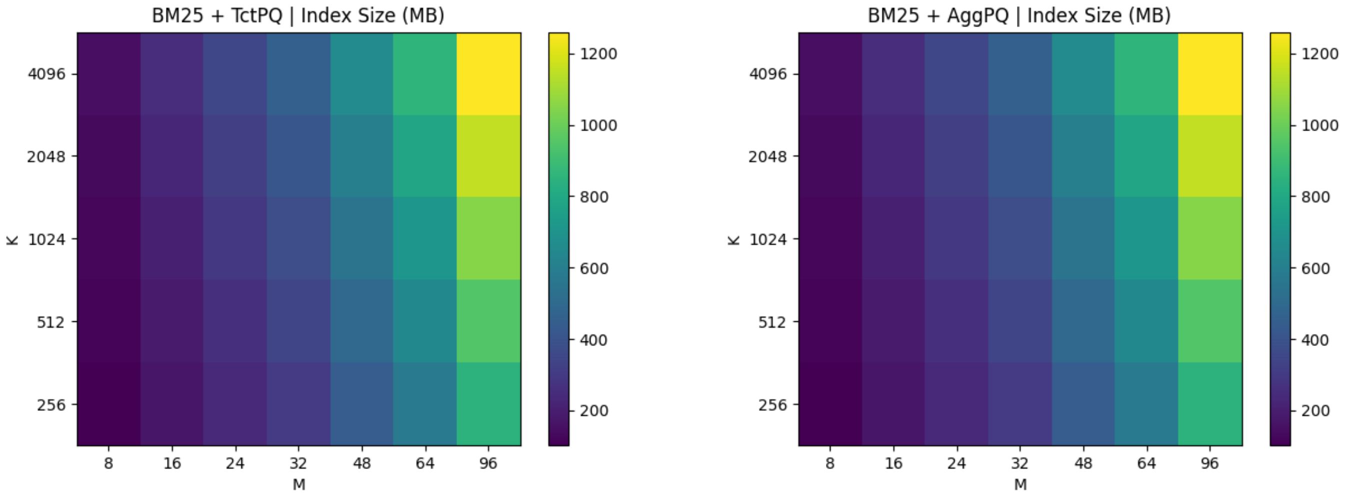


Figure 7.1: The influence of M (number of subspaces) and K (codebook size) on the size of the PQ-quantized index for TCT-ColBERT (left) and Aggretriever (right) on the MSMARCO Passage Corpus.

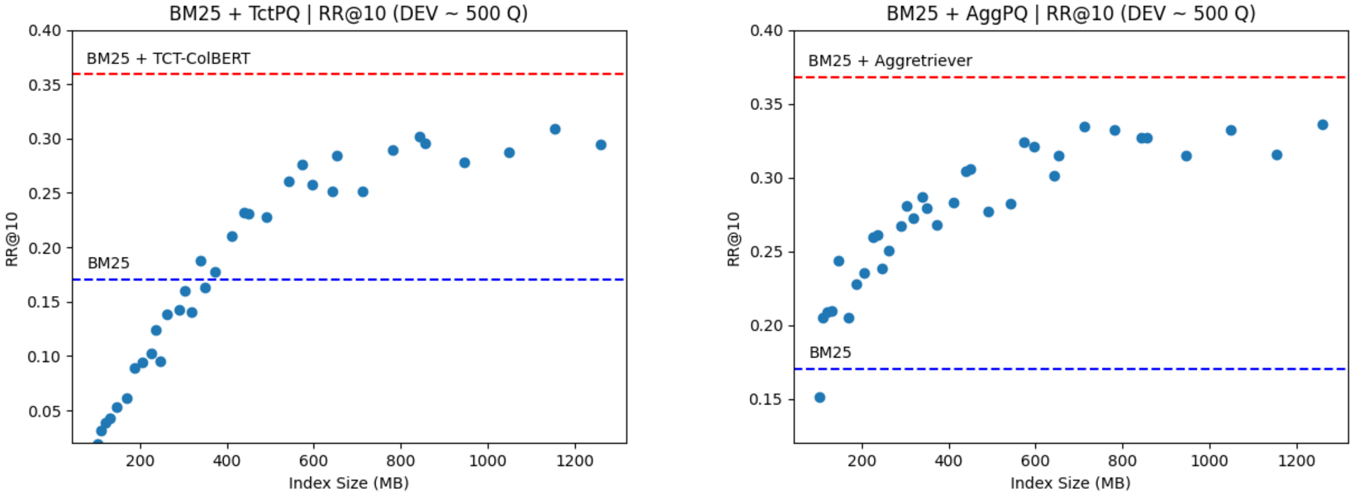


Figure 7.2: Performance of TCT-ColBERT-PQ (left) and Aggretriever-PQ (right) at different index sizes, measured on MSMARCO Passage Dev

7.1.2. Model Comparison

Figure 7.3 illustrates the differences in index size and re-ranking performance between our PQ-quantized single-vector dual-encoders on one side and ColBERT-PQ on the other side. What we can observe is that while the compressed TCT-ColBERT consistently performs worse than the compressed ColBERT index, the quantized Aggretriver outperforms ColBERT-PQ on the 2019 test dataset.

Having said that, the difference in size between the quantized single-vector indices and the ColBERT-PQ index is stark. While the sizes of the former are around 1.2GB for the chosen quantization parameters, the latter index has a size of roughly 10.2GB, which is x10 more.

In contrast, as indicated in the tables A.2 and A.3, even our worse performing PQ-quantized index achieves re-ranking scores, which are only roughly 10% lower than ColBERT-PQ. And, as already mentioned, Aggretriever-PQ outperforms ColBERT-PQ on the 2019 dataset, and it obtains an NDCG@10 score that is only 2% worse on the 2020 test dataset.

Thus, we can firmly claim that re-ranking using quantized *single-vector* dual encoders achieves a better memory-performance trade-off than approaches relying on compressed token-level dual-encoders, like the one introduced in the contextual quantization paper [70].

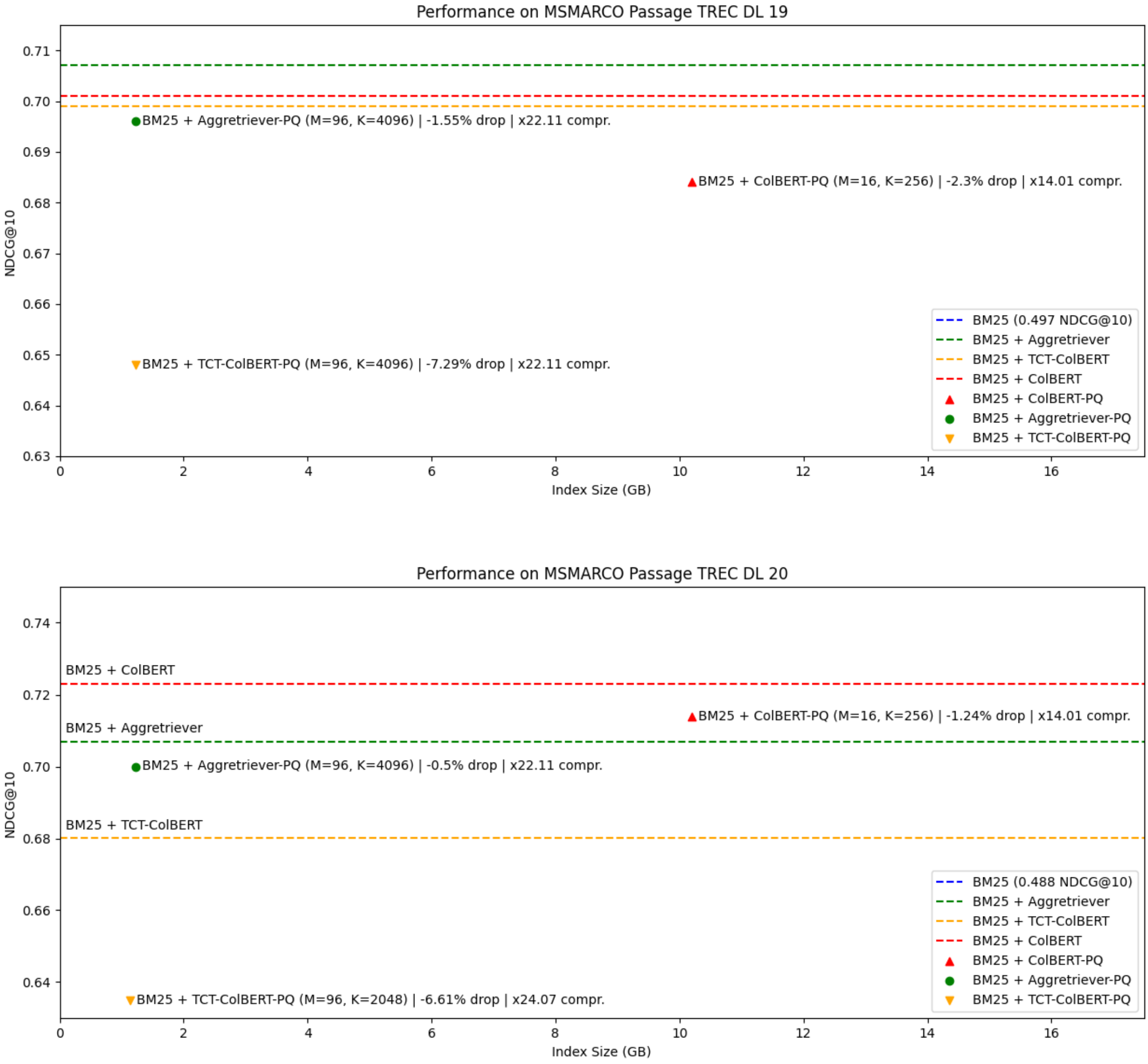


Figure 7.3: Diagrams illustrating the memory-performance differences trade-off differences between BM25+Aggretriever-PQ, BM25+TCT-ColBERT-PQ and BM25+ColBERT-PQ. The index size is measured in GB, while the re-ranking performance uses NDCG@10. The diagram above shows performance measured on the 2019 test dataset, while the one below relies on the more recent 2020 dataset. The dotted lines indicate baseline performances of non-quantized versions of the setup.

7.2. Unsupervized quantization with interpolation

- **RQ2**: Can we increase the compression rate of product quantization, while maintaining the re-ranking performance, by introducing interpolation with sparse scores to the re-ranking score function?

An approach similar to the one for the first research question has been taken to answer the second research question. Specifically, we performed a set of experiments that mirrors the ones done for the first question. This time, the system used in the experiments included the sparse-score interpolation augmentation presented in chapter 5. The results for interpolated PQ-quantized re-ranking were compared against the results shown in the previous section.

7.2.1. Alpha optimization

We performed experiments on the dev set to determine the optimal sparse-interpolation parameter α (alpha) for both quantized dual-encoder re-rankers. Figure 7.4 illustrates the performers of the re-rankers at various values of α. Optimum performance is achieved for $\alpha = 0.3$ in the case of Aggretriever-PQ and $\alpha = 0.1$ for TCT-ColBERT-PQ. The optimum values of α are the same across all three evaluation metrics, as illustrated by figure 7.4.

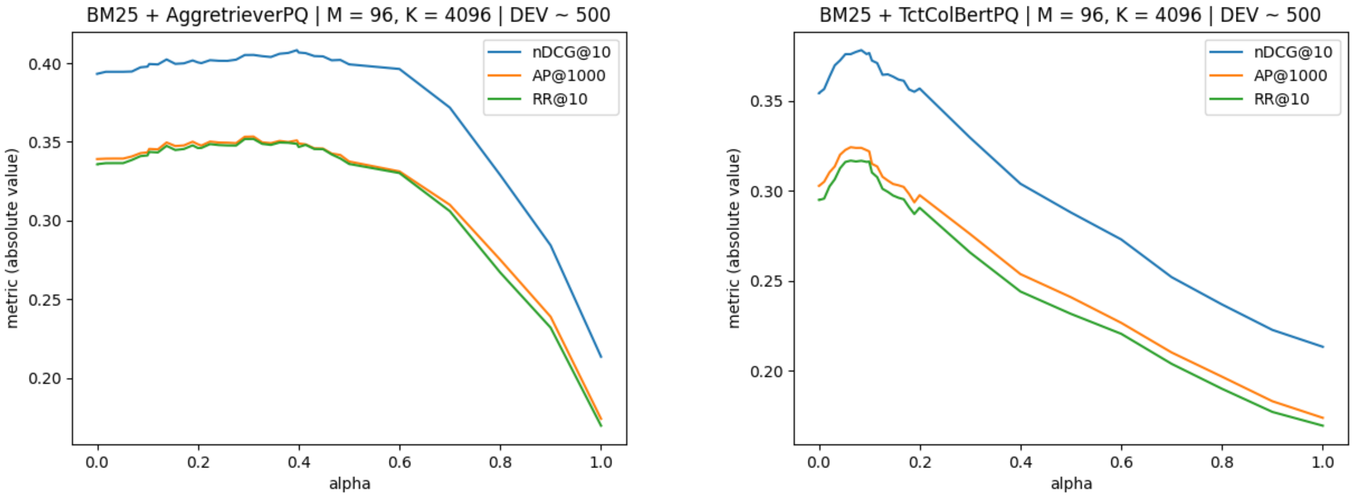


Figure 7.4: Diagram illustrating the re-ranking performance of BM25 + Aggretriever-PQ (left) and BM25 + TCT-ColBERT-PQ (right) at various values of α.

One useful aspect to notice is that, despite the fact that the same optimum α holds for both quantized and non-quantized indices, the behaviour of the two differs as we vary α. This is illustrated by figure 7.5, which illustrates the performance gained by applying sparse score interpolation to both quantized and non-quantized indices. What we can see is that, across both dual-encoder models, the quantized dual-encoders gain more from interpolation than their uncompressed counterparts. This insight is one of the things that contributed to the idea of training WolfPQ using a listwise loss function.

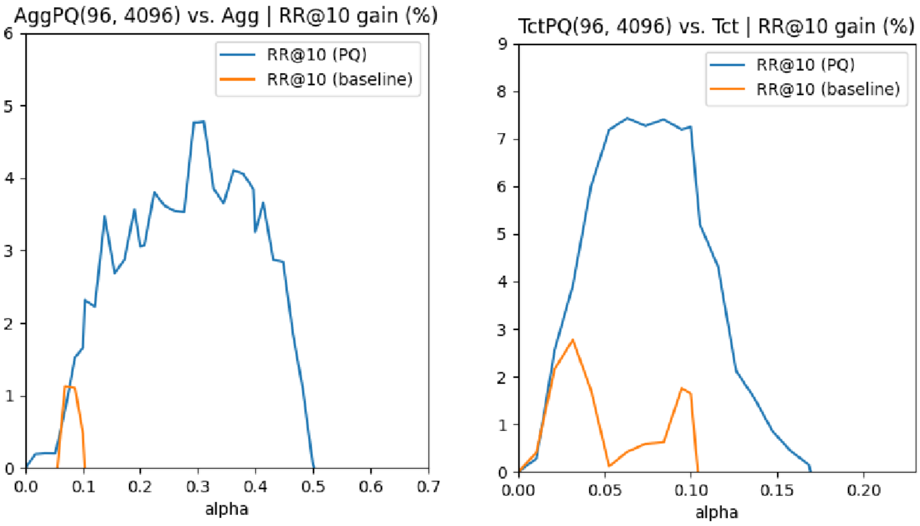


Figure 7.5: Diagram illustrating how much the re-ranking performance of BM25 + Aggretriever-PQ (left) and BM25 + TCT-ColBERT-PQ (right) gains from interpolation at various values of α.

7.2.2. Analysis of quantization with interpolation

In an analogous fashion to figure 7.2, figure 7.6 illustrates the re-ranking performance of the quantized dual-encoder indices with respect to their size for both the interpolated ($\alpha > 0$) and non-interpolated ($\alpha = 0$) cases. One aspect that we can notice is that the interpolated quantized indices manifest the same asymptotic convergence towards the original score of their non-quantized counterparts. What's more important, however, is the fact that at low index sizes (high compression rates), the quantized indices benefit hugely from interpolation. This is especially visible in the case of TCT-ColBERT, which, in cases of very high degrees of compression, can make jumps of over 200% just by leveraging sparse-score interpolation, as shown in table A.4.

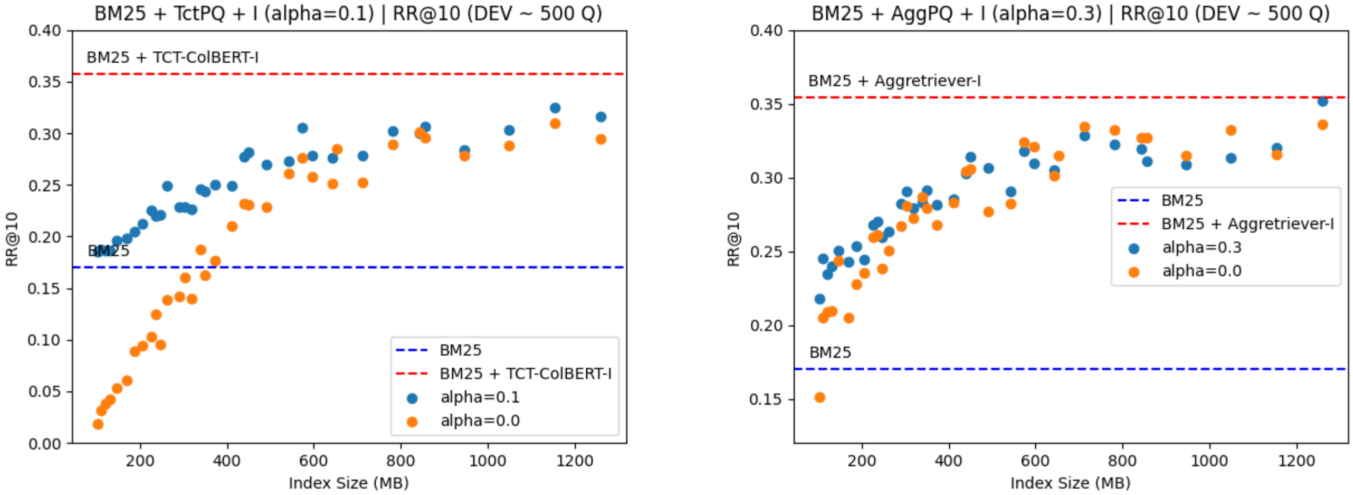


Figure 7.6: Diagram showing the re-ranking performance of BM25 + TCT-ColBERT-PQ(-I) (left) and BM25 + Aggretriever-PQ(-I) (right) at various index sizes (compression rates) with and without interpolation.

The core insight is that the jumps in performance seem to always bring quantized indices that were performing worse than bare BM25 above the BM25 line. This is very important since re-ranking systems that perform under the BM25 baseline are useless, as they can just be substituted by plain BM25. Hence, sparse-score interpolation can be leveraged to perform extreme levels of compression while still keeping the compressed dense index useful.

Another way to look at things is that sparse score interpolation can be used to obtain a more compressed index, provided that you want to remain within a certain re-ranking performance range. This is illustrated by figures 7.7 and 7.8, which highlight how much smaller an interpolated quantized dense index can be to obtain a similar or higher performance than the plain non-interpolated index. For instance, a 170MB quantized TCT-ColBERT interpolated index achieves a similar re-ranking performance as a 339MB non-interpolated quantized index, which means that interpolation enables us to reduce the size of the index by almost two times. However, as shown by the diagrams, as we move into the space of larger, less compressed quantized indices, the positive effects of sparse-score interpolation diminish.

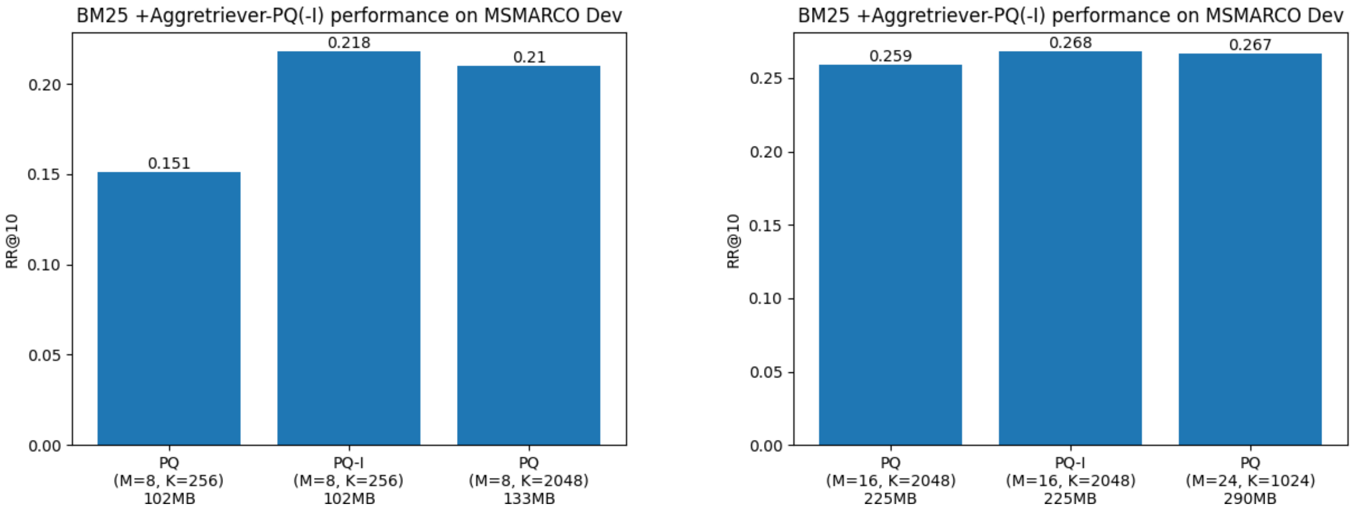


Figure 7.7: Re-ranking performance of BM25+Aggretriever-PQ(-I) measured on MSMARCO Dev, with and without interpolation, at various index sized (compression rates).

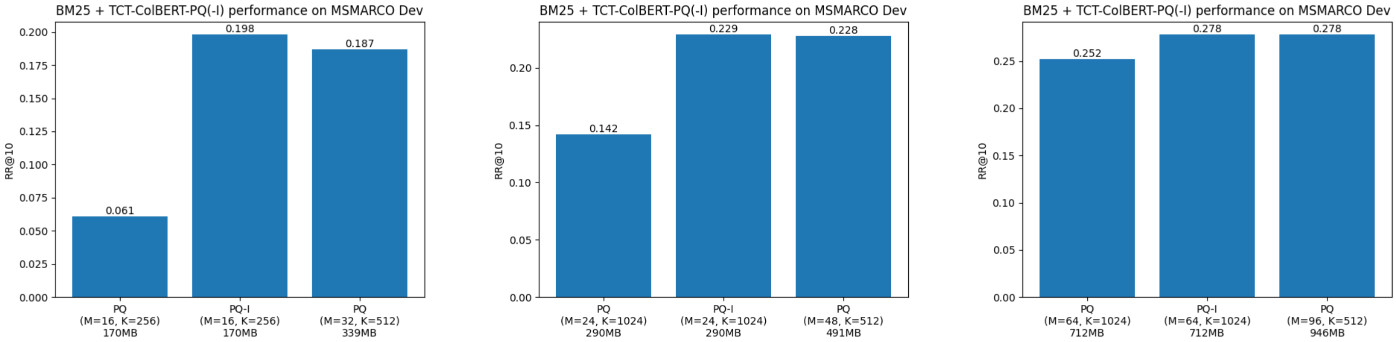
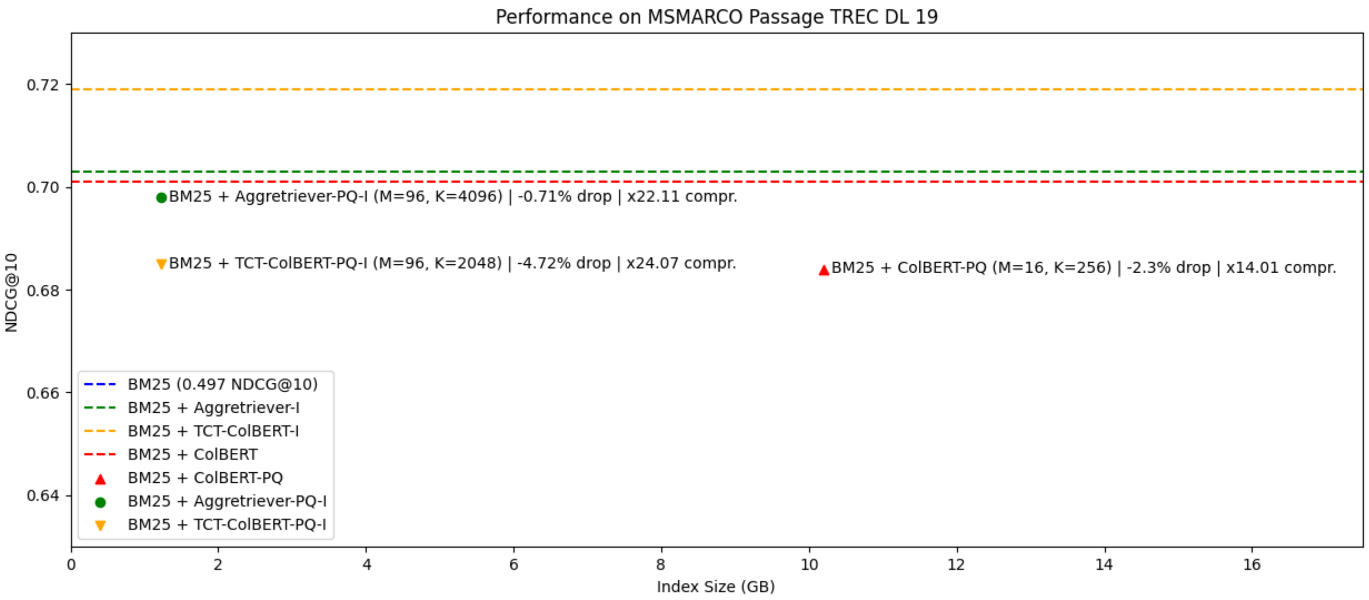


Figure 7.8: Re-ranking performance of BM25+TCT-ColBERT-PQ(-I) measured on MSMARCO Dev, with and without interpolation, at various index sized (compression rates).

7.2.3. Model comparison

The diagrams in figure 7.9 showcase the NDCG@10 score of the two of our best-performing PQ-quantized sparse-score interpolated indices. As opposed to the non-interpolated case illustrated in figure 7.3, we can notice a very high jump in the performance of the TCT-ColBERT index. On the 2019 dataset, both re-rankers perform better than their ColBERT counterpart. Having said this, since sparse-score interpolation has no impact on the dense index size, the observations made in the previous subsection hold. Specifically, the single-vector quantized re-rankers achieve a better performance-memory tradeoff than their ColBERT counterpart. This is even more true in the case of the interpolated re-rankers since interpolation leads to higher performances for *free*. While it would have been interesting to see a comparison with the interpolated ColBERT index, the huge size of ColBERT made such experiments unfeasible, provided our available hardware, time and cost limitations.



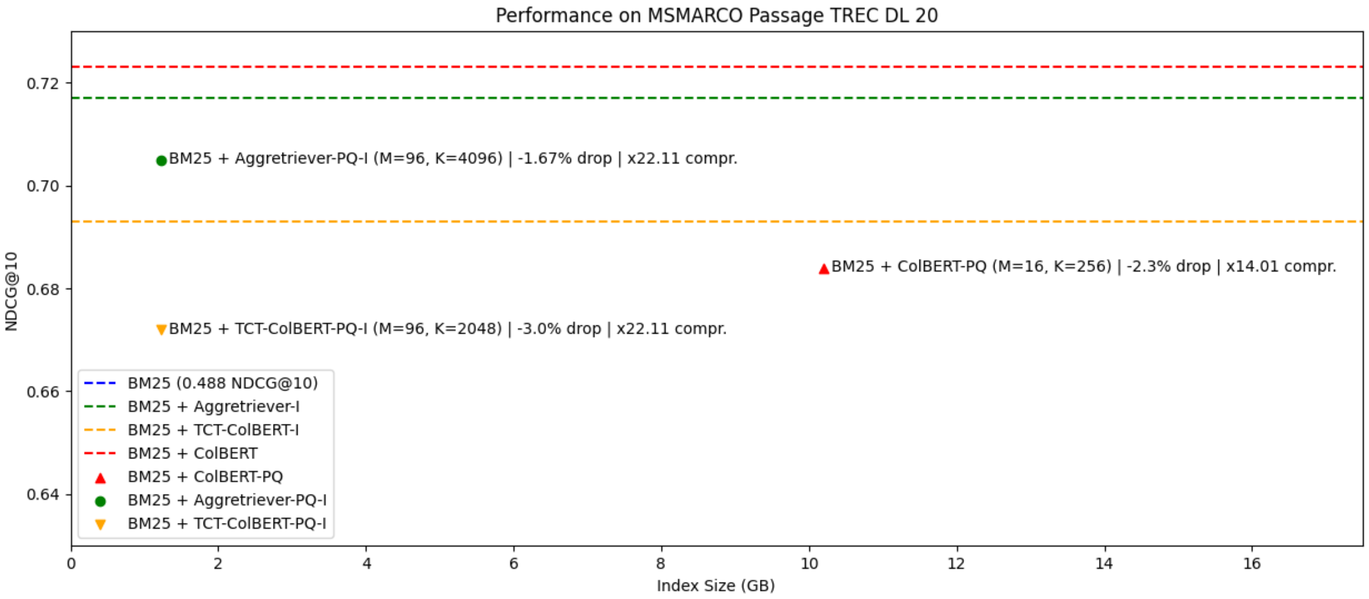


Figure 7.9: Diagrams illustrating the memory-performance differences trade-off differences between BM25+Aggretriever-PQ-I, BM25+TCT-ColBERT-PQ-I and BM25+ColBERT-PQ. The index size is measured in GB, while the re-ranking performance uses NDCG@10. The diagram above shows performance measured on the 2019 test dataset, while the one below relies on the more recent 2020 dataset. The dotted lines indicate baseline performances of non-quantized versions of the setup.

7.3. Supervized quantization for reranking

- **RQ3**: Can we train a quantizer in a supervised fashion to optimize its behaviour for dense re-ranking? If so, can we train the quantizer in a way that enables it to adapt to sparse score interpolation achieving even higher performance improvements over product quantization?

As a way to answer the last research question, many experiments were performed on WolfPQ, the novel supervised quantizer for re-ranking that this paper proposes. This involved pre-training and fine-tuning many configurations of WolfPQ and then evaluating the performance of the full quantized re-ranking system on the MSMARCO dataset. Since training WolfPQ is a lengthy process, the experiments focused on three configurations corresponding to high, medium and low compression rate settings. In addition to that, only the final models were trained on the full dataset. Hyperparameter optimization was done on partially trained models.

7.3.1. Pre-training

Figure 7.10 highlights the performance of WolfPQ after only pre-training as compared to the performance of PQ. While developing the model, one of the aims was to achieve similar results to PQ after the pre-training phase. The intuition behind this aim was that supervised fine-tuning would undoubtedly improve re-ranking performance to a certain degree. Hence, starting the fine-tuning process with a model that already performs similarly to PQ will increase the chances of the final model outperforming PQ. As the diagram shows, the aim was achieved since WolfPQ has a re-ranking performance that's only slightly worse for the high and medium levels of compression and slightly better for low compression.

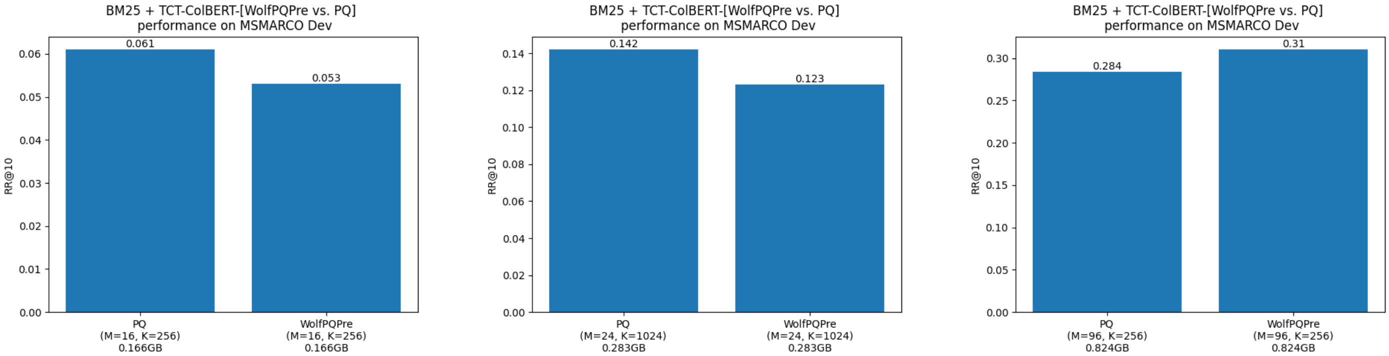


Figure 7.10: Re-ranking performance of BM25+TCT-ColBERT-WolfPQPre (WolfPQ after only pre-training) at various index size, i.e. 0.166GB (left), 0.283GB (middle), 0.824GB (right)

7.3.2. Listwise loss vs. pointwise loss

As already mentioned in chapter 5, one of the design trade-offs that went into WolfPQ was choosing to train it using the *MarginMSE* [32] loss function (pointwise loss) or *RankNet* [8] (listwise loss). The trade-off is resolved by introducing LIP, a parameter that controls the linear interpolation between the two. The impact of this training parameter in WolfPQ is explored in figure 7.11.

The left diagram of figure 7.11 corresponds to instances of WolfPQ trained using fully dense (non-interpolated scores). In contrast, the right diagram shows the performance of WolfPQ trained using sparse-interpolated scores (abbreviated WolfPQI). What we can notice is that for the fully dense training case, the optimum value for the LIP parameter is 0, which is translated to training the model using $MarginMSE$ only. In contrast, when training WolfPQ with interpolated scores, the performance peaks at $LIP = 0.7$, suggesting that the listwise loss function is beneficial.

The experimental results confirm our design intuition that listwise training is useful in getting the quantizer to adapt to sparse interpolation. However, leaving sparse interpolation aside, *MarginMSE* seems to be the superior loss function, as indicated by the fact that for $LIP = 1$ (full listwise training), the model always underperforms.

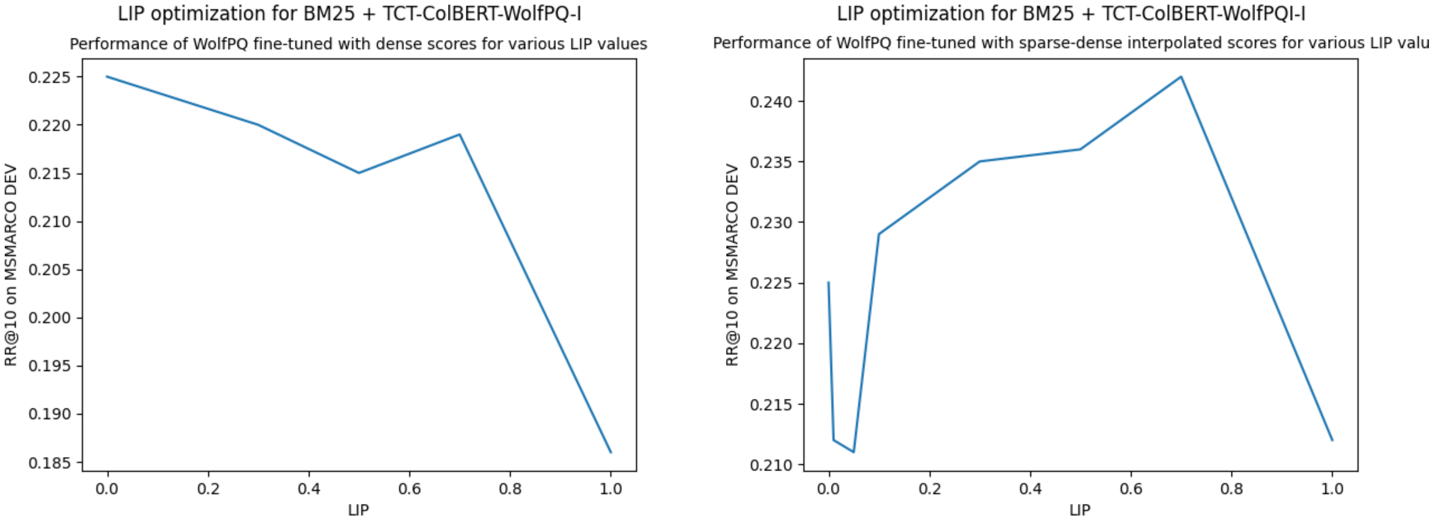


Figure 7.11: Re-ranking performance of BM25 + TCT-ColBERT-WolfPQ-I (left, trained without sparse interpolation) and BM25 + TCT-ColBERT-WolfPQI-I (right, trained with sparse interpolation) for versions of WolfPQ trained using various values for LIP (loss interpolation parameter)

7.3.3. Exploration of semantic sampling

Another core design trade-off involved in WolfPQ is the decision between *selection by minimum distance* and *selection by semantic sampling*, both techniques having been described in great detail in chapter 5. As already mentioned in the chapter, selection by semantic sampling still relies on minimum distance, whose degree of contribution is controlled by the MDF hyperparameter. Pure semantic sampling would correspond to $MDF = 0$, while pure selection by minimum distance to $MDF \to \infty$ (conceptually only, as it is, of course, not implemented like that).

Figure 7.12 showcases the re-ranking performance of WolfPQ with selection by semantic sampling (abbreviated WolfPQSem) for various values of MDF. As we can see, pure semantic sampling performs poorly. This confirms our intuition that selection by minimum distance is already a good baseline strategy and that the optimal selection mechanism does not differ too much from it. As we increase MDF, so does the re-ranking performance of WolfPQSem. The performance peaks close to $MDF = 30$, after which it approaches the baseline performance of selection by minimum distance asymptotically, indicated by the red line. This matches the intuition that semantic sampling with $MDF \to \infty$ is essentially selection by minimum distance.

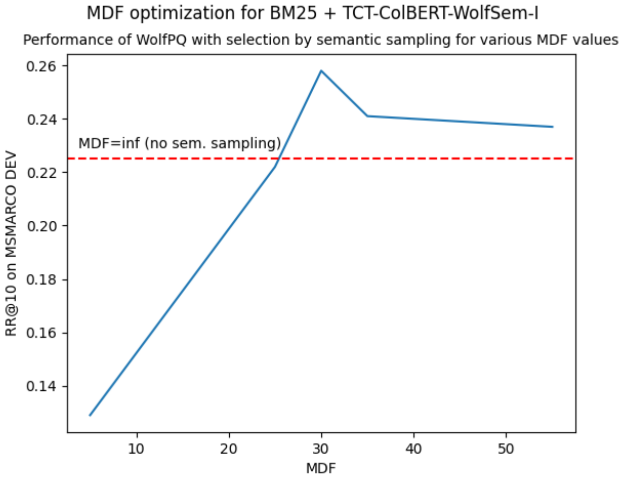


Figure 7.12: Re-ranking performance of BM25 + TctColBERT-WolfPQSem-I trained with various values for the MDF parameter.

7.3.4. WolfPQ compared to PQ

Having determined the optimal parameters of WolfPQ on the dev set, a few final configurations were tested on TREC DL 2019 and TREC DL 2020. The abbreviations and explanations are listed below. The suffix *-I* indicates that re-ranking with interpolation was done to measure the performance.

- **WolfPQ** - classical version of WolfPQ that uses selection by minimum distance and was trained using fully dense scores and $LIP = 0$ (purely pointwise loss).
- **WolfPQSem** - same as WolfPQ, but using selection by minimum distance with $MDF = 30$
- **WolfPQI** - same as WolfPQ, but trained using sparse-interpolated scores and $LIP = 0.7$ (mixed pointwise-listwise loss training)
- **WolfPQSemI** - same as WolfPQI, but using selection by minimum distance with $MDF = 30$

For WolfPQI and WolfPQSemI only the results of applying re-ranking with interpolation are reported. The reason is that it does not make sense to train the models using sparse-scores and then avoid applying sparse interpolation during inference.

Figures 7.13, 7.14 and 7.15 showcase the performance of the various configurations of WolfPQ as compared to classical PQ on the 2019 and 2020 test sets for the three levels of compression, high, medium and low. One of the first aspects to notice is that irrespective of the test set and level of compression, WolfPQ outperforms PQ in the case of non-interpolated re-ranking. This conforms to our hypothesis that there is a goal mismatch between quantization for approximate nearest neighbour search (ANN), for which unsupervised quantizers like PQ are close to optimum, and quantization for re-ranking. As indicated by the experimental results, the goal mismatch can be mitigated by training a vector quantizer in a supervised way, specifically on the re-ranking problem.

Another insight that we can get by looking at the results in the two figures is that supervised quantization induces higher performance gains over PQ at a higher level of compression since, for lower compression, normal PQ already performs quite well. For instance, on TREC DL 19, at the medium compression level (index size of 0.283GB), WolfPQ outperforms PQ by roughly 19%. In contrast, at the low compression setting, the gain of WolfPQ over PQ is only 3%. This insight follows a common trend that we have encountered in the research so far, i.e. improvements of quantized re-ranking systems have most of the impact at very high degrees of compression, since for very low degrees of compression, any decent algorithm will do acceptably well.

Moving on to the results of re-ranking with interpolation (the blue bars), we notice that sparse interpolation levels the playing field between PQ and WolfPQ to a certain degree. In other words, poor-performing quantizers, like PQ, gain more from interpolation than higher-performing quantizers. While WolfPQ generally maintains its superiority over PQ in the interpolated setting, the discrepancy between the two becomes much lower. In addition to that, certain configurations of WolfPQ are sometimes outperformed by PQ once interpolation comes into play. For instance, in the high compression setting, PQ-I outperforms WolfPQ-I by 5.1% despite performing much worse without interpolation. The best result for that setting is achieved by WolfPQI-I, which is trained to adapt to an interpolated setting.

The inconsistency between which configuration works best once interpolation comes into play, regardless of the non-interpolated results, suggests that a large part of the differences in re-ranking performance between the interpolated quantizers can be accounted for by the stochasticity of the hyperparameter initialization and training. Hence, while WolfPQ might learn to adapt to interpolation, as some of our best results come from the interpolated training setting, there seems to be a lot of room for improvement when it comes to this aspect.

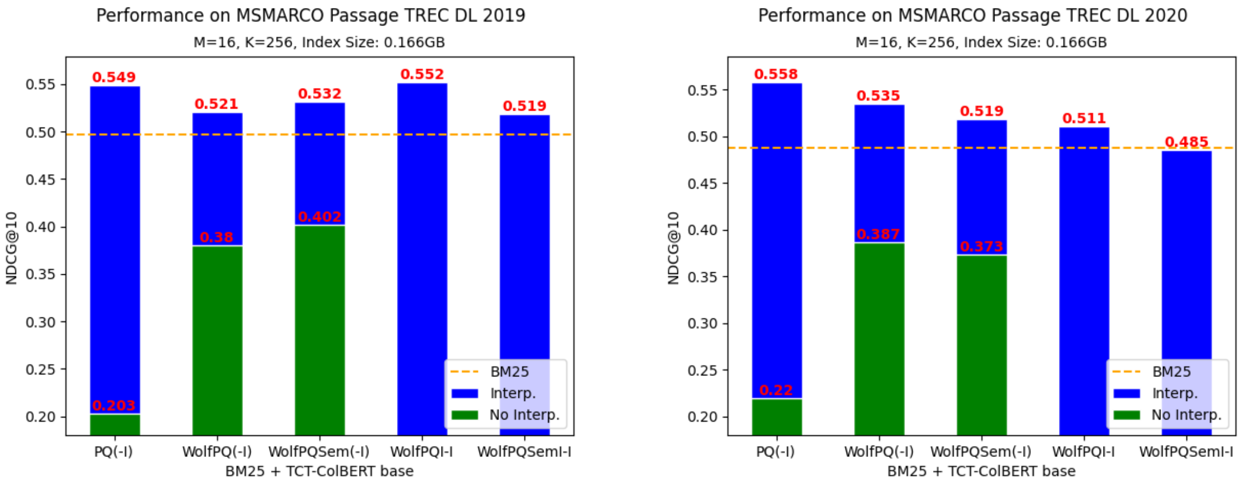


Figure 7.13: Comparison between various configurations of WolfPQ and PQ, on the 2019 test set (left) and on the 2020 test set (right), at a small index size, i.e. 0.166GB

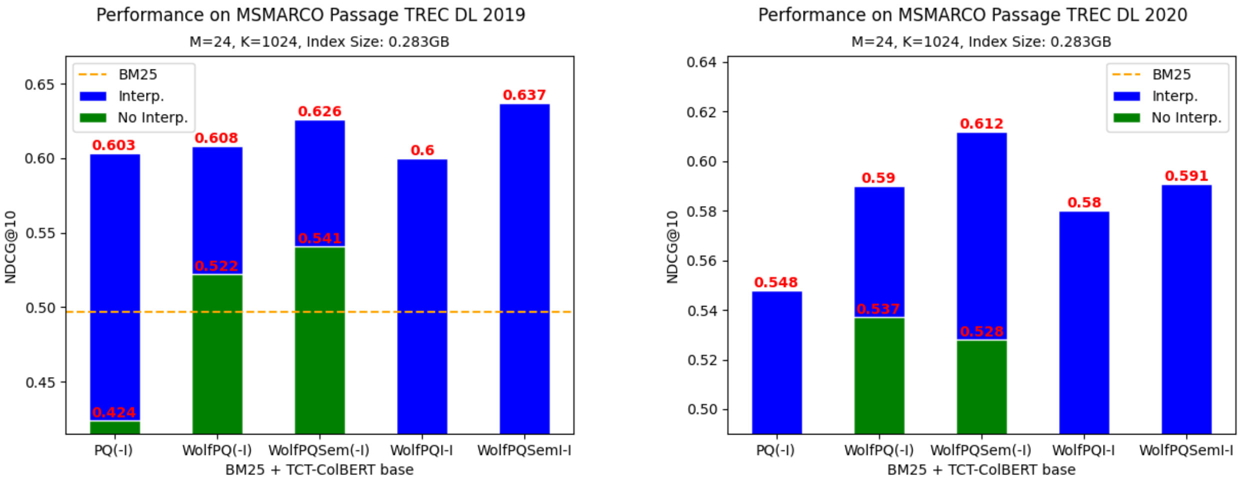


Figure 7.14: Comparison between various configurations of WolfPQ and PQ, on the 2019 test set (left) and on the 2020 test set (right), at a medium index size, i.e. 0.283GB

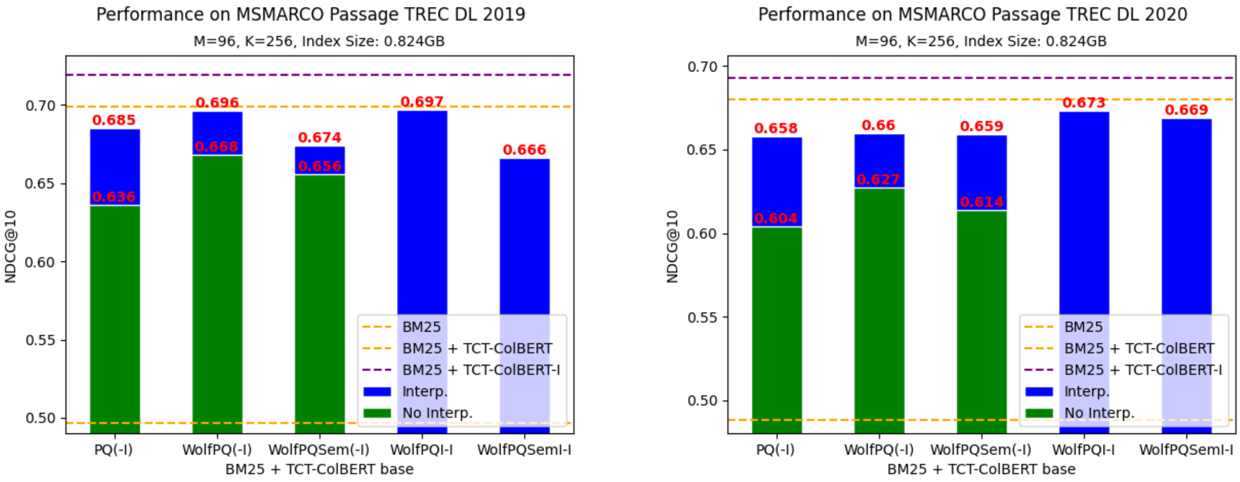


Figure 7.15: Comparison between various configurations of WolfPQ and PQ, on the 2019 test set (left) and on the 2020 test set (right), at a large index size, i.e. 0.824GB

7.3.5. Comparison to other systems

Having seen how WolfPQ compares to classical PQ in both the non-interpolated and interpolated settings, it is time to look into how our best performing model compares to other methods found in the literature, which were taken as baselines.

Figure 7.16 illustrates the performance of our sparse retrieval dense re-ranking system quantized with the best configuration of WolfPQ (*BM25 + TCT-ColBERT-WolfPQI-I*) compared to other retrieval systems.

First of all, one of the first aspects to notice is that our WolfPQ-based system is slightly outperformed by our main competitor, which combines ColBERT-based re-ranking together with contextual quantization (BM25 + ColBERT-CQ). As shown in table A.13, BM25 + ColBERT-CQ achieves an NDCG@10 score of 0.704 on TREC DL 2019, while our model achieves 0.697, which is 1% worse. Having said that, the ColBERT-CQ index has a size of 10.2GB, while our WolfPQ-based index is only 0.824GB, which is x12.37 less. Thus, we can firmly claim that our proposed solution achieves a better performance-memory trade-off. While the difference between the performance of the two systems is higher on the TREC DL 2020 dataset (roughly 6%), the large difference in memory size would likely still make our solution more appealing for a plethora of memory-sensitive applications.

In addition to the above, WolfPQ manages to outperform JPQ [22], a close to state-of-the-art quantized dense retrieval system, by a significant margin. For a similar index size of 0.83GB, our WolfPQ-based setup achieves an NDCG@10 score of 0.697, while JPQ achieves only 0.677, which is a 2.9% difference. Unlike JPQ, the WolfPQ-based setup also has the advantage of not having to retrain the query encoder, which might not always be possible due to a lack of computational resources or cost.

When compared to RepCONC [72], the state-of-the-art quantized dense retrieval model, our method achieves higher NDCG@10 performance but at the downside of a slightly higher index size (0.393GB vs 0.824GB). Since the NDCG@10 performance of RepCONC at our exact index sizes is not reported in its paper, we cannot firmly say which of the two setups achieves the best performance-memory trade-off. However, RepCONC has certain disadvantages that would still make our solution more appealing in many situations. On the one hand, RepCONC involves the joint training of a full dual-encoder together with product quantization (PQ), which requires a lot of processing power and large datasets. On the other hand, RepCONC is a dense retrieval system, which is likely to be slower than our BM25 sparse retrieval, given a large enough corpus. In addition to all of these, WolfPQ can be applied to any new dual-encoder that might become state-of-the-art in the future, as our solution is decoupled from the dense index.

Finally, the diagram in figure 7.16 and table 7.1 highlight the big performance difference between sparse-retrieval combined with quantized dense re-ranking and quantized dense-retrieval using plain PQ. Hence, it can be claimed that sparse retrieval quantized re-ranking is likely a superior setup than quantized retrieval.

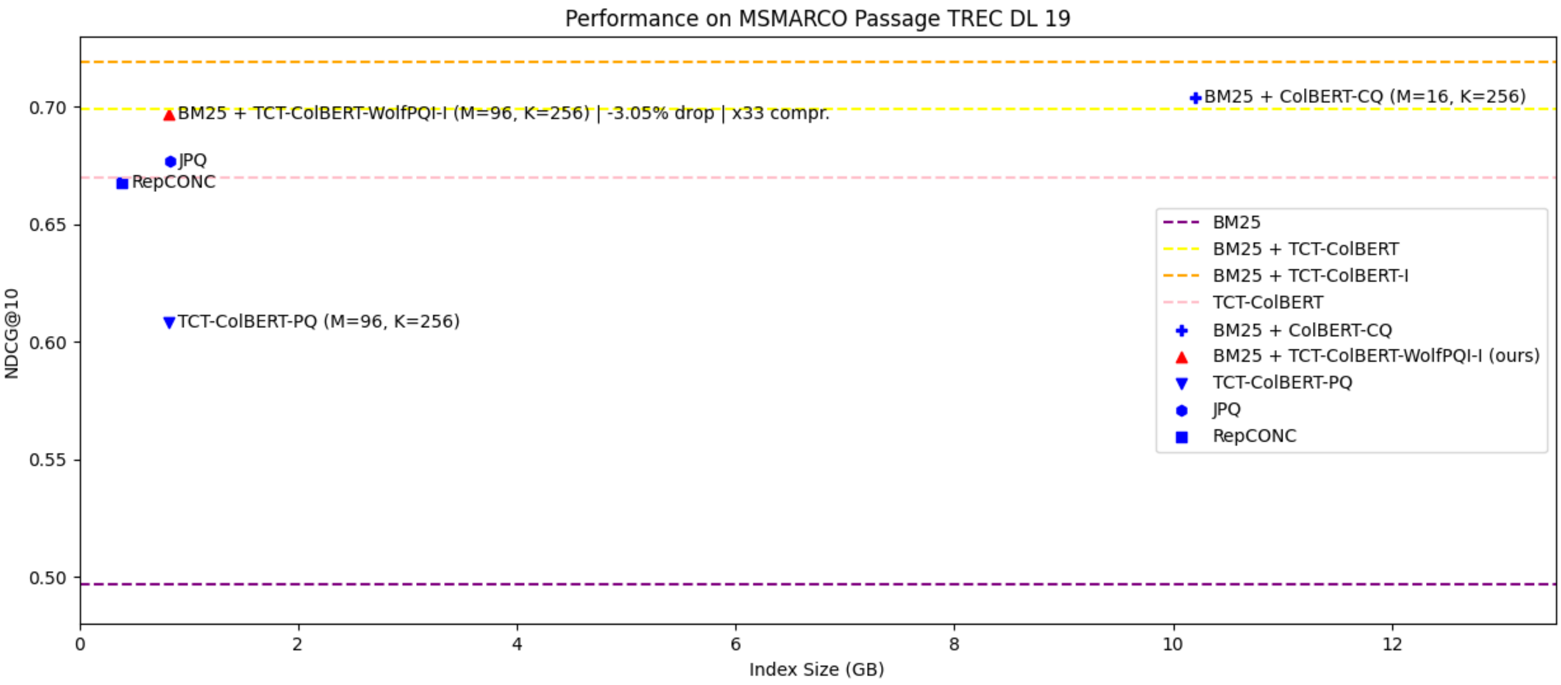


Figure 7.16: Final comparison between the best WolfPQ configuration (BM25 + TCT-ColBERT-WolfPQI-I) and relevant baselines, measured on the 2019 test set.

Model	M	K	Index Size (GB)	NDCG@10 (2019)	NDCG@10 (2020)
BM25	-	-	-	0.497	0.488
TCT-ColBERT	-	-	27.2	0.670*	-
JPQ	-	-	0.83*	0.677*	-
RepCONC	-	-	0.393*	0.668*	-
BM25 + TCT-ColBERT	-	-	27.2	0.699	0.680
BM25 + TCT-ColBERT-I	-	-	27.2	0.719	0.693
BM25 + ColBERT-CQ	16	256	10.2*	0.704*	0.716*
TCT-ColBERT-PQ	96	256	0.824	0.608	0.544
BM25 + TCT-ColBERT-WolfPQI-I	96	256	0.824	**0.697**	**0.673**

Table 7.1: Re-ranking performence on MSMARCO Passage TREC DL 2019 and 2020

8

Conclusion and future work

The purpose of this chapter is provide the reader with the conclusions of the research, summarizing the contributions, discussing the limitations of the proposed methods, as well as speculating about possible future work. To this end, the first section will go over the answers to the research questions presented in relation to the methodology and the results. Building on that, the second section will discuss limitations of WolfPQ. Finally, the third section will provide a brief commentary on how the research could be continued.

8.1. Research conclusions

As mentioned above, we will go over the answers to the questions that have been at the core of this research and thesis. The section is structured in three subsections, each presenting the findings and contributions related to one of the research questions listed below.

- **RQ1**: How well does product quantization work with single-vector dual encoders, specifically TCT-ColBERT and Aggretriever?
- **RQ2**: Can we increase the compression rate of product quantization, while maintaining the re-ranking performance, by introducing interpolation with sparse scores to the reranking score function?
- **RQ3**: Can we train a quantizer in a supervised fashion to optimize its behaviour for dense re-ranking? If so, can we train the quantizer in a way that enables it to adapt to sparse score interpolation achieving even higher performance improvements over product quantization?

8.1.1. Research question 1

In the previous chapter, we have seen that to answer the first research question, the two targeted single-vector dual encoders (TCT-ColBERT [47] and Aggretriever [46]) have been compressed using various configurations of PQ [36]. The results have been plotted and analyzed, as well as compared with the results of ColBERT-PQ experiments done in the contextual quantization paper [70].

What we have observed is that, at the cost of slightly worse re-ranking performance, TCT-ColBERT-PQ and Aggretriever-PQ use roughly x10 less memory than ColBERT-PQ. Hence, it can be concluded that PQ-quantized single-vector dual-encoders achieve a *better memory-performance trade-off* than their multi-vector counterparts. Thus, we can firmly say that PQ achieves promising results when applied to TCT-ColBERT and Aggretriver in the context of re-ranking.

8.1.2. Research question 2

To answer the second research question a similar experimental design has been used as for the first research question. The same compressed single-vector dual encoder indices obtained by applying PQ have been evaluated in a setting in which re-ranking is done by applying sparse-interpolation.

The outcomes of the experiments have shown that gains of up to 224.6% can be achieved by leveraging sparse interpolation. The gains obtained from sparse interpolation are the largest at very high compression ratios and tend to decrease as the compression ratio becomes lower. Regardless of that, the experimental data has shown that the intersection of sparse-score interpolation with PQ is an extremely powerful technique that can make the difference between a highly compressed re-ranking index being viable (having performance above the BM25 baseline) or not. Thus, it can be affirmed with a high degree of certainty that sparse-score interpolation is a technique with no drawbacks that can be used to compensate to a great extent for the performance drop resulting from applying PQ.

In addition to the above, the thesis has highlighted that quantization with sparse interpolation is a highly effective way to maintain the re-ranking performance above a certain threshold while decreasing the memory usage of the dense index as much as possible. For instance, a 339MB PQ-quantized TCT-ColBERT index can be substituted with an equivalent 170MB index that uses sparse interpolation. Hence, we can conclude that the answer to the second research question is definitely positive.

8.1.3. Research question 3

The main contribution of the thesis consists of having answered the third and last research question. To explore whether or not a supervised quantizer can be trained to optimize re-ranking, our thesis proposes WolfPQ.

As seen in chapter 5, WolfPQ is a novel quantization technique that is trained in an end-to-end setting with the purpose of adapting quantization to re-ranking. This was achieved by adopting a design methodology with the aim of mitigating the objective gap between the training of unsupervised quantizers like PQ and the goals of passage re-ranking. Multiple configurations and training procedures for WolfPQ have been tried and tested with the objective of obtaining a quantizer that achieves higher interpolation gains. In addition to that, two codeword selection methods have been explored, i.e. *selection by minimum distance* and *selection by semantic sampling*.

The results of the experiments have shown that WolfPQ is able to outperform PQ by a large margin in the non-interpolated setting. The differences in performance between the two quantization methods tend to be larger at high degrees of compression. The higher performance of WolfPQ when compared to PQ shows that the before-mentioned objective gap can be mitigated by training a quantizer specifically for the task of passage re-ranking.

When it comes to the design choice between selection by minimum distance and selection by semantic sampling, the final results are inconclusive. Selection by semantic sampling seems to perform better on the 2019 test set, while the minimum distance approach is superior on the 2020 dataset. Further research would be necessary to tell with certainty which approach is the best or if the choice between the two is a hyperparameter that needs to be tuned on each individual training dataset.

When introducing sparse score interpolation, the performance difference between PQ and WolfPQ is reduced. In other words, quantizers with a worse base performance tend to benefit more from sparse interpolation. As one would expect, configurations of WolfPQ trained using sparse-interpolated scores and a mixed pointwise-listwise loss function tend to outperform the other configurations once interpolation comes into play. Having said that, this does not happen under all settings and datasets.

The great variability when it comes to which quantization configuration performs best in a setting involving sparse-interpolation could be explained by the stochasticity of initialization and training. It also suggests that there is room for improvement when it comes to how well WolfPQ adapts to interpolation.

When comparing our best WolfPQ configuration with representative baselines such as JPQ [22], BM25 + ColBERT-CQ [70] or dense retrieval by TCT-ColBERT-PQ, it becomes apparent that WolfPQ manages to achieve a better memory-performance trade-off.

In contrast to the above, further experiments are needed to determine with certainty whether or not RepCONC [72], a state-of-the-art quantized retrieval model, is inferior or superior to WolfPQ in terms of memory-performance trade-off. Despite this, we can firmly claim that WolfPQ has certain usability advantages over RepCONC. On the one hand, RepCONC, unlike WolfPQ, requires the joint training of a dual-encoder and product quantizer, which might not be feasible due to either lack of a large enough dataset, computational resources or cost. On the other hand, RepCONC relies on dense retrieval, rather than sparse-retrieval and dense-reranking, as our proposed setup, which is likely to be slower given large enough corpora [44]. Finally, WolfPQ is completely decoupled from the dual-encoder, which means that it has the potential to be applied to future improved dense ranking models.

Taking all of the above into account, one could argue that WolfPQ is a strong candidate for a state-of-the-art technique. However more experiments would be necessary to determine that for certain.

When it comes to answering the third research question, we can state with confidence that supervised quantization is a viable way to achieve a superior performance-memory trade-off in a quantized single-vector dual-encoder. When it comes to training the quantizer to better adapt to sparse-interpolation, further research would be necessary. However, our slight performance increases resulting from our training with sparse-interpolated scores, as well as the variability of results once interpolation is added, provides some evidence that getting a quantizer to leverage interpolation to a higher degree is within the realm of possibility.

8.2. Limitations of the proposed techniques

The main limitation of WolfPQ is the lack of training scalability with respect to the parameters M (the number of subspace divisions) and K (the codebook size). As a consequence of the fact that in WolfPQ, the codebook is trainable, the number of trainable parameters explodes for high values of M and K. This makes it difficult to train WolfPQ for configurations involving lower levels of compression, as GPUs with a lot of RAM are needed.

Another noteworthy limitation is pre-processing the training data. As WolfPQ relies on BM25 negative sampling, preparing the training data in the context of large corpora might induce some time overhead. Of course, this is highly dependent on the available hardware.

Finally, as already established, there is plenty of room for improvement when it comes to the ability of WolfPQ to adapt to sparse-score interpolation in order to benefit from higher performance gains than the PQ baseline. Tackling this limitation would be key to developing even better quantization methods for re-ranking.

8.3. Future work

A clear general direction for future work would be tackling the limitations of WolfPQ. An idea that could be leveraged to improve training scalability would be to avoid training a full quantizer. As our experiments have shown, a relatively good re-ranking performance can already be achieved by applying the simple PQ algorithm. Hence, one solution that has potential is to train an adapter on top of PQ. To be more specific, one could train a neural network to perform a transformation of the quantized vectors outputted by PQ to make them more prone towards giving high re-ranking performance. The architecture of such a neural network would not be dependent on the parameters of PQ (M and K) but only on the dimension of the semantic vectors.

As mentioned a couple of times already, a promising direction for improving quantization for re-ranking would be finding a way to learn a quantizer that can leverage interpolation better. One way of achieving this could be the development of a new loss function that takes into account the vocabulary mismatch between documents. In essence, since documents with greater vocabulary mismatch will likely have different BM25 scores, the dense scores are more important in distinguishing between documents with greater vocabulary similarity. Hence, it is important to make it less likely for the quantizer to assign documents with very close vocabulary similarity to the same codeword.

An alternative to the idea presented above would be the use of the BM25 scores directly as input data into the quantizer in the hope that the quantizer would learn a better assignment strategy that takes the scores into account.

Another research direction would be exploring the application of compression in a non-uniform manner across the semantic space. As of right now, the entire semantic space is compressed using the same configuration of the quantizer. However, since certain areas of the semantic space are more likely to be retrieved than others, it makes sense to apply more compression to the less popular regions and less compression to more popular regions. In this way, we minimize the information lost about the important documents in the corpus at the expense of losing information about less important documents.

References

[1] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. “The k-means algorithm: A comprehensive survey and performance evaluation”. In: *Electronics* 9.8 (2020), p. 1295.

[2] Perplexity AI. *How does perplexity work?* 2024. url: `https://www.perplexity.ai/hub/faq/how-does-perplexity-work` (visited on 06/14/2024).

[3] Akiko Aizawa. “An information-theoretic perspective of tf–idf measures”. In: *Information Processing & Management* 39.1 (2003), pp. 45–65.

[4] Artem Babenko and Victor Lempitsky. “Additive quantization for extreme vector compression”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 931–938.

[5] Payal Bajaj et al. *MS MARCO: A Human Generated MAchine Reading COmprehension Dataset*. 2018. arXiv: `1611.09268`.

[6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation learning: A review and new perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[7] Antal van den Bosch, Toine Bogers, and Maurice de Kunder. “Estimating search engine index size variability: a 9-year longitudinal study”. In: *Scientometrics* 107.2 (May 2016), pp. 839–856. issn: 1588-2861. doi: `10.1007/s11192-016-1863-z`. url: `https://doi.org/10.1007/s11192-016-1863-z`.

[8] Chris Burges et al. “Learning to rank using gradient descent”. In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 89–96.

[9] Christopher Burges, Robert Ragno, and Quoc Le. “Learning to rank with nonsmooth cost functions”. In: *Advances in neural information processing systems* 19 (2006).

[10] Luis Cabrera-Cordon. *Introducing semantic search: Bringing more meaningful results to Azure Cognitive Search*. 2021. url: `https://techcommunity.microsoft.com/t5/ai-azure-ai-services-blog/introducing-semantic-search-bringing-more-meaningful-results-to/ba-p/2175636` (visited on 03/02/2021).

[11] Daniel Campos et al. *MS MARCO*. 2024. url: `https://microsoft.github.io/msmarco/` (visited on 06/16/2024).

[12] Shuangshuang Chen and Wei Guo. “Auto-encoders in deep learning—a review with new perspectives”. In: *Mathematics* 11.8 (2023), p. 1777.

[13] Ting Chen, Lala Li, and Yizhou Sun. “Differentiable Product Quantization for End-to-End Embedding Compression”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 1617–1626. url: `https://proceedings.mlr.press/v119/chen20l.html`.

[14] Houssem-Eddine Chihoub et al. “Exploring energy-consistency trade-offs in cassandra cloud storage system”. In: *2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE. 2015, pp. 146–153.

[15] Wenxiao Chu et al. “Review of Hot Topics in the Sustainable Development of Energy, Water, and Environment Systems Conference in 2022”. In: *Energies* 16.23 (2023), p. 7897.

[16] Lorenzo Ciampiconi et al. “A survey and taxonomy of loss functions in machine learning”. In: *arXiv preprint arXiv:2301.05579* (2023).

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

[18] Nick Craswell et al. “Overview of the TREC 2020 deep learning track”. In: *arXiv e-prints* (2021), arXiv–2102.

[19] *DelftBlue: the TU Delft supercomputer*. 2024. url: `https://www.tudelft.nl/dhpc/system` (visited on 06/16/2024).

[20] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. “Activation functions in deep learning: A comprehensive survey and benchmark”. In: *Neurocomputing* 503 (2022), pp. 92–108. issn: 0925-2312. doi: `https://doi.org/10.1016/j.neucom.2022.06.111`. url: `https://www.sciencedirect.com/science/article/pii/S0925231222008426`.

[21] *Faiss*. 2024. url: `https://ai.meta.com/tools/faiss/` (visited on 06/16/2024).

[22] Yan Fang et al. “Joint Optimization of Multi-vector Representation with Product Quantization”. In: *Natural Language Processing and Chinese Computing*. Ed. by Wei Lu et al. Cham: Springer International Publishing, 2022, pp. 631–642. isbn: 978-3-031-17120-8.

[23] *fast-forward-indexes*. 2024. url: `https://pypi.org/project/fast-forward-indexes/` (visited on 06/16/2024).

[24] Thibault Formal et al. “SPLADE v2: Sparse lexical and expansion model for information retrieval”. In: *arXiv preprint arXiv:2109.10086* (2021).

[25] George W. Furnas et al. “The vocabulary problem in human-system communication”. In: *Communications of the ACM* 30.11 (1987), pp. 964–971.

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

[27] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep sparse rectifier neural networks”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[29] *Google Colab*. 2024. url: `https://colab.research.google.com/` (visited on 06/16/2024).

[30] Jiafeng Guo et al. “A Deep Look into neural ranking models for information retrieval”. In: *Information Processing Management* 57.6 (2020), p. 102067. issn: 0306-4573. doi: `https://doi.org/10.1016/j.ipm.2019.102067`. url: `https://www.sciencedirect.com/science/article/pii/S0306457319302390`.

[31] *HDF5*. 2024. url: `https://www.hdfgroup.org/solutions/hdf5/` (visited on 06/16/2024).

[32] Sebastian Hofstätter et al. “Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation”. In: (2020).

[33] *Hugging Face Transformers*. 2024. url: `https://huggingface.co/docs/transformers/en/index` (visited on 06/16/2024).

[34] *ir-measures*. 2024. url: `https://ir-measur.es/en/latest/` (visited on 06/16/2024).

[35] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical reparameterization with gumbel-softmax”. In: *arXiv preprint arXiv:1611.01144* (2016).

[36] Herve Jégou, Matthijs Douze, and Cordelia Schmid. “Product Quantization for Nearest Neighbor Search”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1 (2011), pp. 117–128. doi: `10.1109/TPAMI.2010.57`.

[37] Euna Jung, Jaekeol Choi, and Wonjong Rhee. “Semi-siamese bi-encoder neural ranking model using lightweight fine-tuning”. In: *Proceedings of the ACM Web Conference 2022*. 2022, pp. 502–511.

[38] Sato Kaz and Shi Guangsha. *Your RAGs powered by Google Search technology, part 2*. 2024. url: `https://cloud.google.com/blog/products/ai-machine-learning/rags-powered-by-google-search-technology-part-2` (visited on 02/14/2024).

[39] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *Proceedings of naacL-HLT*. Vol. 1. 2019, p. 2.

[40] 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*. 2020, pp. 39–48.

[41] Solomon Kullback and Richard A Leibler. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[42] Maurice de Kunder. *The size of the World Wide Web (The Internet)*. 2024. url: `https://worldwidewebsize.com/` (visited on 06/15/2024).

[43] Phong Le and Willem Zuidema. “Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive LSTMs”. In: *arXiv preprint arXiv:1603.00423* (2016).

[44] Jurek Leonhardt et al. “Efficient Neural Ranking using Forward Indexes”. In: *Proceedings of the ACM Web Conference 2022*. WWW ’22. <conf-loc>, <city>Virtual Event, Lyon</city>, <country>France</country>, </conf-loc>: Association for Computing Machinery, 2022, pp. 266–276. isbn: 9781450390965. doi: `10.1145/3485447.3511955`. url: `https://doi.org/10.1145/3485447.3511955`.

[45] Minghong Lin et al. “Online algorithms for geographical load balancing”. In: *2012 international green computing conference (IGCC)*. IEEE. 2012, pp. 1–10.

[46] Sheng-Chieh Lin, Minghan Li, and Jimmy Lin. “Aggretriever: A Simple Approach to Aggregate Textual Representations for Robust Dense Passage Retrieval”. In: *Transactions of the Association for Computational Linguistics* 11 (2023), pp. 436–452.

[47] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. *Distilling Dense Representations for Ranking using Tightly-Coupled Teachers*. 2020. arXiv: `2010.11386`.

[48] *Lucene search*. 2024. url: `https://lucene.apache.org/` (visited on 06/16/2024).

[49] Emil Sauer Lynge. “Recurrent neural networks for language modeling”. In: (2016).

[50] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. “The concrete distribution: A continuous relaxation of discrete random variables”. In: *arXiv preprint arXiv:1611.00712* (2016).

[51] Antonio Mallia et al. “Learning passage impacts for inverted indexes”. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2021, pp. 1723–1727.

[52] Maryamah Maryamah et al. “Chatbots in Academia: A Retrieval-Augmented Generation Approach for Improved Efficient Information Access”. In: *2024 16th International Conference on Knowledge and Smart Technology (KST)*. 2024, pp. 259–264. doi: `10.1109/KST61284.2024.10499652`.

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

[54] Bonan Min et al. “Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey”. In: *ACM Comput. Surv.* 56.2 (Sept. 2023). issn: 0360-0300. doi: `10.1145/3605943`. url: `https://doi.org/10.1145/3605943`.

[55] Zahra Monfared, Jonas Magdy Mikhaeil, and Daniel Durstewitz. “How to train RNNs on chaotic data?” In: (2021).

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

[57] Mohammad Norouzi and David J. Fleet. “Cartesian K-Means”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2013.

[58] Matthew E. Peters et al. “Deep Contextualized Word Representations”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Ed. by Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237. doi: `10.18653/v1/N18-1202`. url: `https://aclanthology.org/N18-1202`.

[59] *PyTorch*. 2024. url: `https://pytorch.org/` (visited on 06/16/2024).

[60] Thilina C Rajapakse, Andrew Yates, and Maarten de Rijke. "Negative Sampling Techniques for Dense Passage Retrieval in a Multilingual Setting". In: (2024).

[61] Stephen Robertson, Hugo Zaragoza, et al. "The probabilistic relevance framework: BM25 and beyond". In: *Foundations and Trends® in Information Retrieval* 3.4 (2009), pp. 333–389.

[62] Hojjat Salehinejad et al. "Recent advances in recurrent neural networks". In: *arXiv preprint arXiv:1801.01078* (2017).

[63] Peter H Schönemann. "A generalized solution of the orthogonal procrustes problem". In: *Psychometrika* 31.1 (1966), pp. 1–10.

[64] Simple Sharma and Supriya P Panda. "Efficient information retrieval model: overcoming challenges in search engines-an overview". In: *Indonesian Journal of Electrical Engineering and Computer Science* 32.2 (2023), pp. 925–932.

[65] Stefano Squartini, Amir Hussain, and Francesco Piazza. "Preprocessing based solution for the vanishing gradient problem in recurrent neural networks". In: *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS'03.* Vol. 5. IEEE. 2003, pp. V–V.

[66] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[67] Jingdong Wang and Ting Zhang. "Composite Quantization". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.6 (2019), pp. 1308–1322. doi: 10.1109/TPAMI.2018.2835468.

[68] 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*. 2021, pp. 317–324.

[69] Lanling Xu et al. *Negative Sampling for Contrastive Representation Learning: A Review*. 2022. arXiv: 2206.00212.

[70] Yingrui Yang, Yifan Qiao, and Tao Yang. "Compact Token Representations with Contextual Quantization for Efficient Document Re-ranking". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*. 2022.

[71] Andrew Yates, Rodrigo Nogueira, and Jimmy Lin. "Pretrained Transformers for Text Ranking: BERT and Beyond". In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM '21. Virtual Event, Israel: Association for Computing Machinery, 2021, pp. 1154–1156. isbn: 9781450382977. doi: 10.1145/3437963.3441667. url: https://doi.org/10.1145/3437963.3441667.

[72] Jingtao Zhan et al. "Learning Discrete Representations via Constrained Clustering for Effective and Efficient Dense Retrieval". In: *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. WSDM '22. Virtual Event, AZ, USA: Association for Computing Machinery, 2022, pp. 1328–1336. isbn: 9781450391320. doi: 10.1145/3488560.3498443. url: https://doi.org/10.1145/3488560.3498443.

[73] Qin Zhang et al. "A survey for efficient open domain question answering". In: *arXiv preprint arXiv:2211.07886* (2022).

[74] Xu Zhang et al. "Correlation encoder-decoder model for text generation". In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2022, pp. 1–7.

[75] Le Zhao and Jamie Callan. "Term necessity prediction". In: *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*. CIKM '10. Toronto, ON, Canada: Association for Computing Machinery, 2010, pp. 259–268. isbn: 9781450300995. doi: 10.1145/1871437.1871474. url: https://doi.org/10.1145/1871437.1871474.

[76] Wayne Xin Zhao et al. "Dense Text Retrieval Based on Pretrained Language Models: A Survey". In: *ACM Trans. Inf. Syst.* 42.4 (Feb. 2024). issn: 1046-8188. doi: 10.1145/3637870. url: https://doi.org/10.1145/3637870.

A

Raw & extra results

A.1. Unsupervised quantization

A.1.1. Hyperparameter exploration

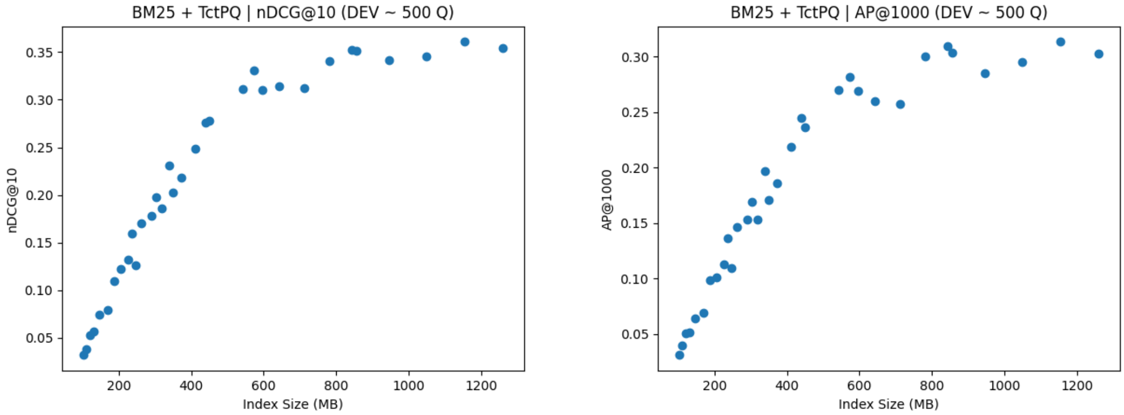


Figure A.1: Performance of TCT-ColBERT-PQ at different index sizes, measured on MSMARCO Passage Dev

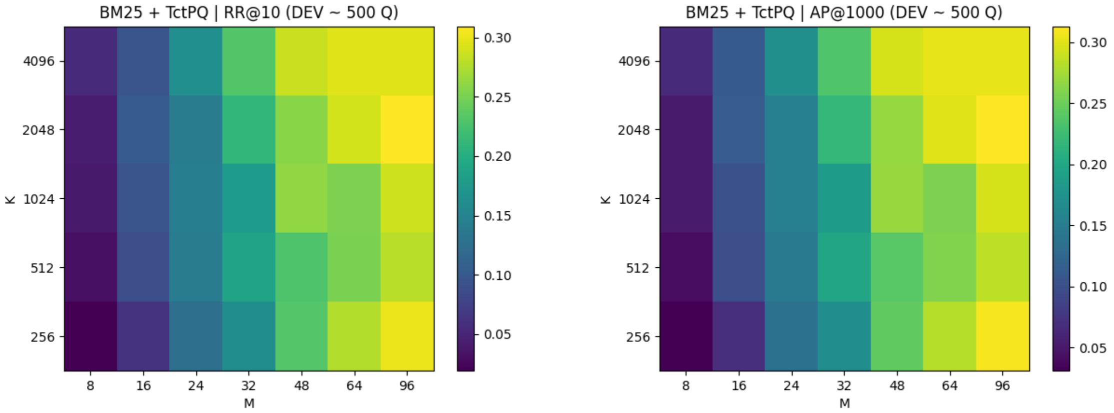


Figure A.2: Performance of TCT-ColBERT-PQ at different index sizes, measured on MSMARCO Passage Dev

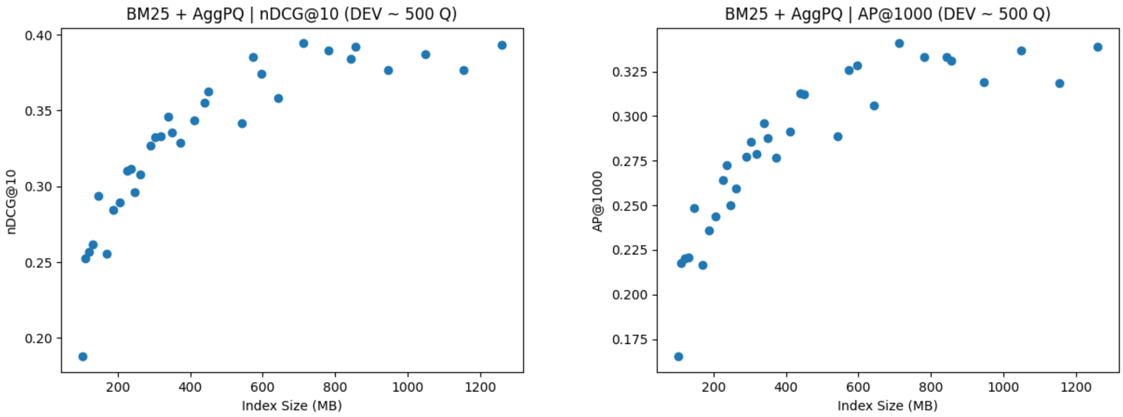


Figure A.3: Performance of TCT-ColBERT-PQ at different index sizes, measured on MSMARCO Passage Dev

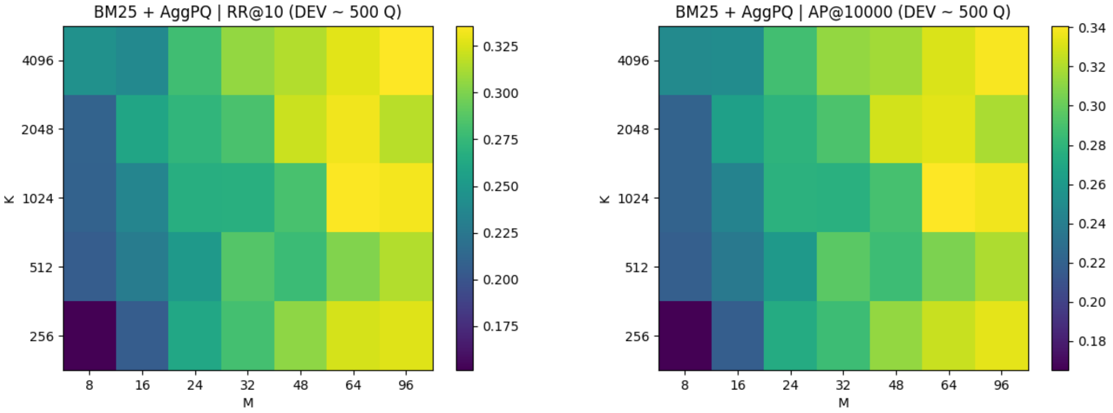
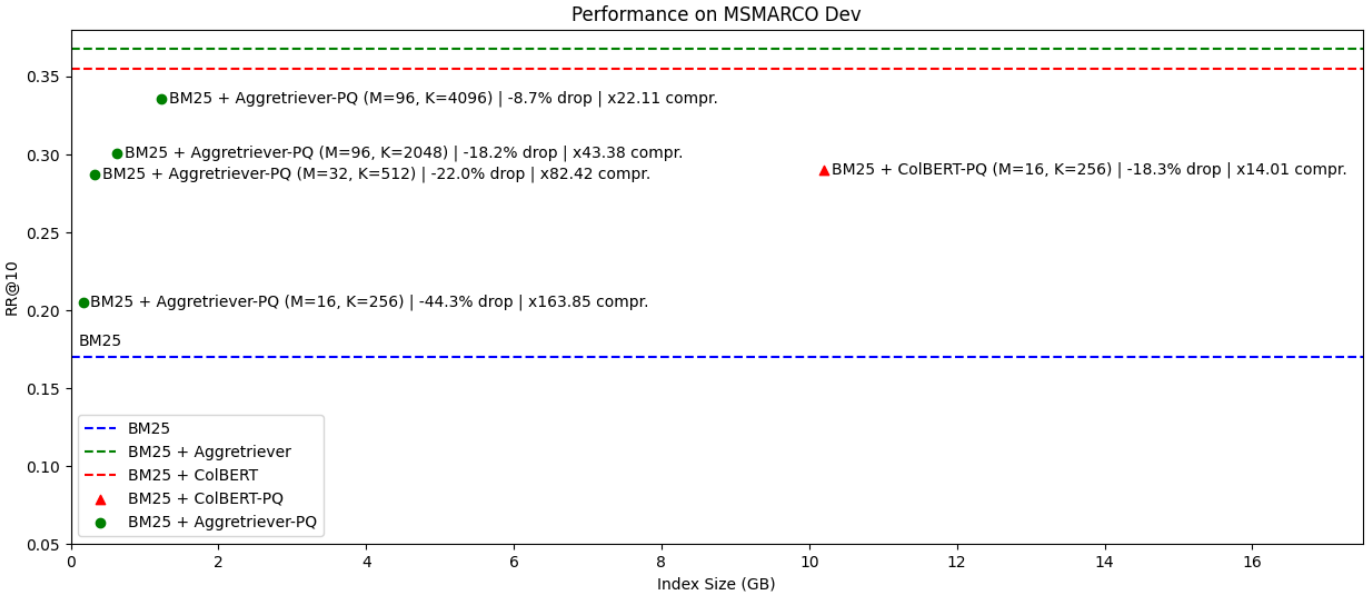


Figure A.4: Performance of TCT-ColBERT-PQ at different index sizes, measured on MSMARCO Passage Dev

A.1.2. Model comparison

Model	M	K	Index Size (GB)	Compression	RR@10	Drop
BM25 (sparse retrieval)	-	-	-	-	0.170	-
BM25 + ColBERT	-	-	143*	-	0.355*	-
BM25 + TCT-ColBERT	-	-	27.2	-	0.359	-
BM25 + Aggretriever	-	-	27.2	-	0.368	-
BM25 + ColBERT-PQ	16	256	10.2*	x14.01	0.290*	-18.3%
BM25 + TCT-ColBERT-PQ	16	256	0.166	x163.85	0.061	-83.0%
BM25 + Aggretriever-PQ	16	256	0.166	x163.85	0.205	-44.3%
BM25 + TCT-ColBERT-PQ	64	2048	0.763	x36.64	0.290	-19.2%
BM25 + Aggretriever-PQ	32	512	0.330	x82.42	0.287	-22.0%
BM25 + TCT-ColBERT-PQ	64	4096	0.834	x32.61	0.296	-17.54%
BM25 + Aggretriever-PQ	64	512	0.627	x43.38	0.301	-18.2%
BM25 + TCT-ColBERT-PQ	96	2048	1.13	x24.07	0.31	-13.6%
BM25 + Aggretriever-PQ	96	4096	1.23	x22.11	0.336	-8.7%

Table A.1: Reranking performence on MSMARCO Passage Dev (rounded to 3 decimals) (RR@10 used, since 1 judgement label per query is too sparse for nDCG@10)



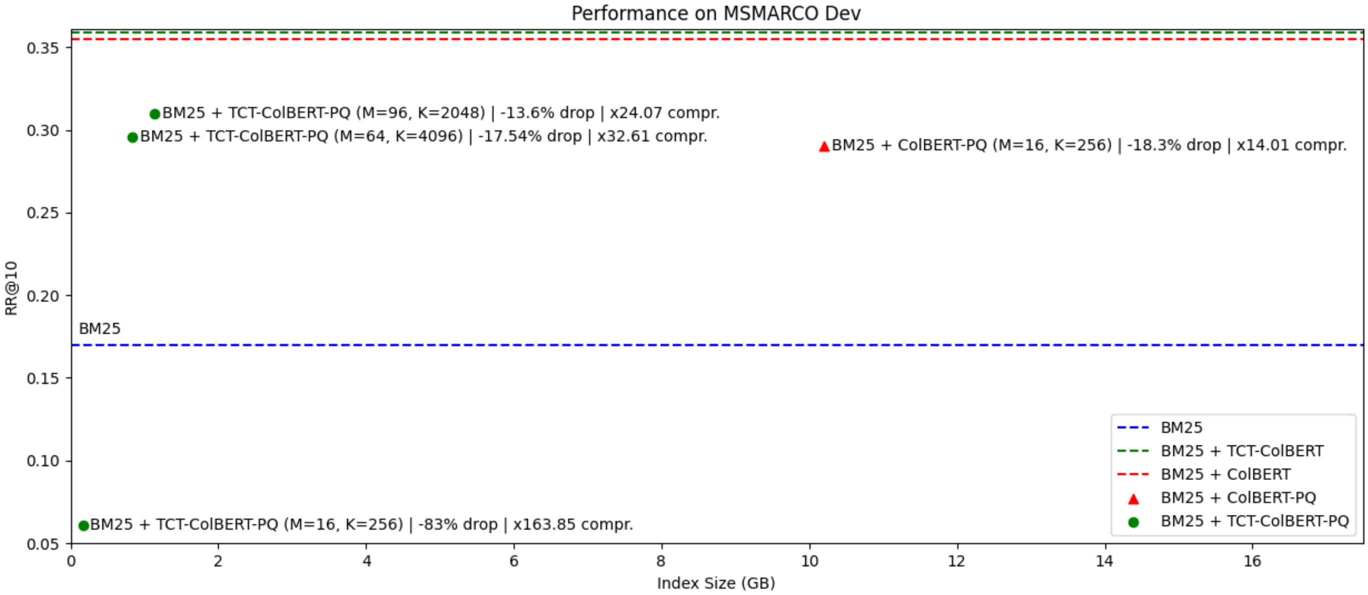


Figure A.5: Diagrams illustrating the memory-performance differences trade-off differences between BM25+Aggretriever-PQ, BM25+TCT-ColBERT-PQ and BM25+ColBERT-PQ. The index size is measured in GB, while the re-ranking performance uses RR@10. The performance is measured on the dev set. The dotted lines indicate baseline performances of non-quantized versions of the setup

Model	M	K	Index Size (GB)	Compression	NDCG@10	Drop
BM25	-	-	-	-	0.497	-
BM25 + ColBERT	-	-	143*	-	0.701*	-
BM25 + TCT-ColBERT	-	-	27.2	-	0.699	-
BM25 + Aggretriever	-	-	27.2	-	0.707	-
BM25 + ColBERT-PQ	16	256	10.2*	x14.01	0.684*	-2.3%
BM25 + TCT-ColBERT-PQ	96	2048	1.13	x24.07	0.636	-9.0%
BM25 + TCT-ColBERT-PQ	96	4096	1.23	x22.11	0.648	-7.29%
BM25 + Aggretriever-PQ	96	4096	1.23	x22.11	0.696	-1.55%

Table A.2: Reranking performence on MSMARCO Passage TREC DL19

Model	M	K	Index Size (GB)	Compression	NDCG@10	Drop
BM25	-	-	-	-	0.488	-
BM25 + ColBERT	-	-	143*	-	0.723*	-
BM25 + TCT-ColBERT	-	-	27.2	-	0.680	-
BM25 + Aggretriever	-	-	27.2	-	0.704	-
BM25 + ColBERT-PQ	16	256	10.2*	x14.01	0.714*	-1.24%
BM25 + TCT-ColBERT-PQ	96	2048	1.13	x24.07	0.635	-6.61%
BM25 + TCT-ColBERT-PQ	96	4096	1.23	x22.11	0.621	-8.67%
BM25 + Aggretriever-PQ	96	4096	1.23	x22.11	0.7	-0.5%

Table A.3: Reranking performence on MSMARCO Passage TREC DL 2020

A.2. Unsupervised quantization with interpolation

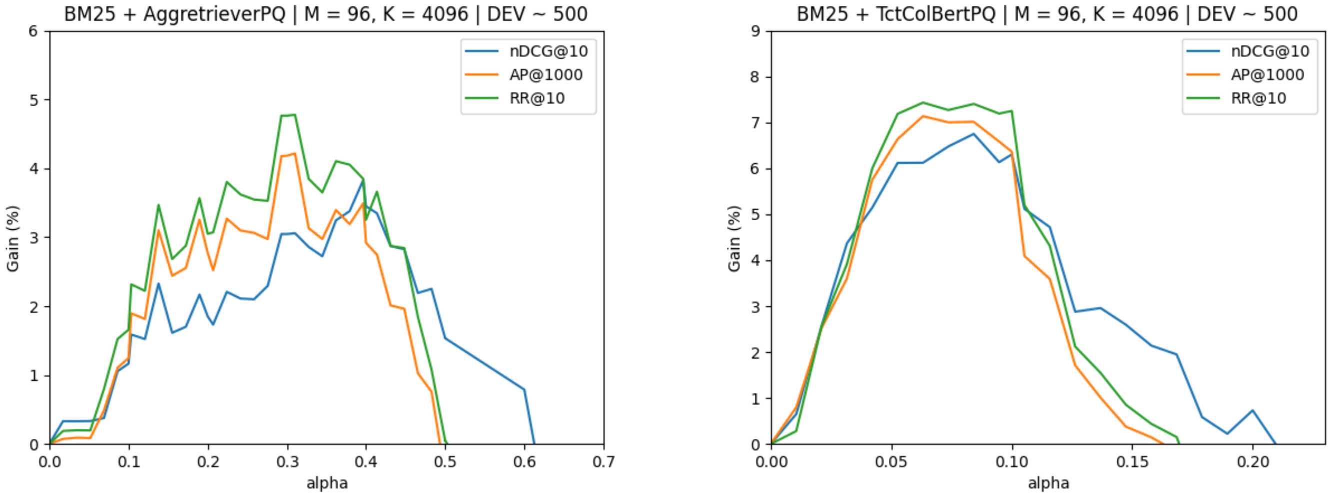


Figure A.6: Diagram illustrating how much the re-ranking performance of BM25 + Aggretriever-PQ-I (left) and BM25 + TCT-ColBERT-PQ-I (rights) gains from interpolation at different values of α.

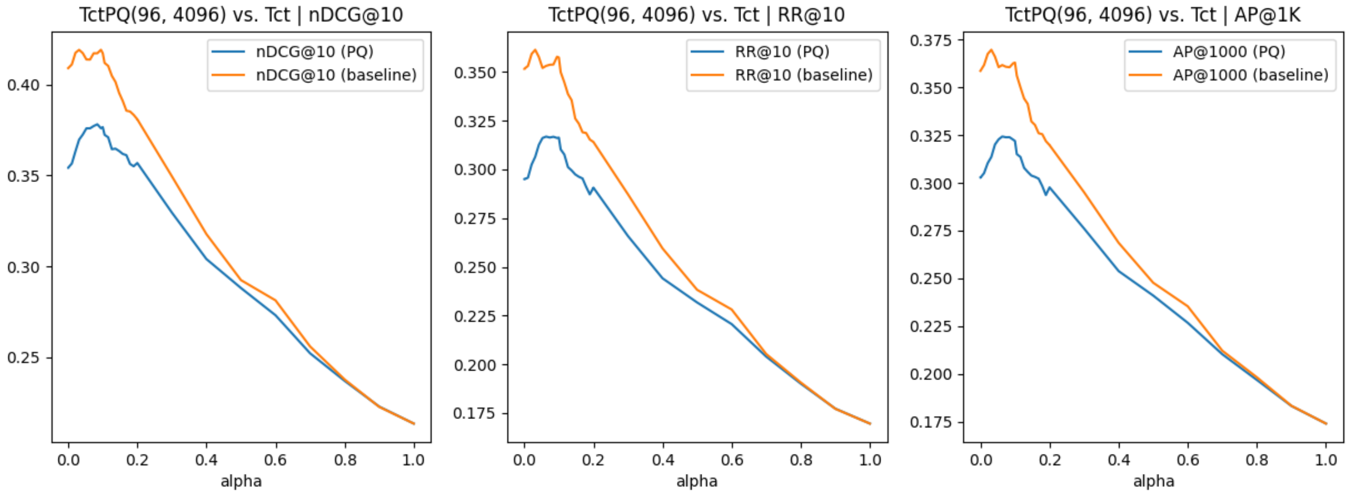


Figure A.7: Performance comparison between TCT-ColBERT and TCT-ColBERT-PQ, with BM25 interpolated scores, at various values of alpha, measured on MSMARCO Passage Dev

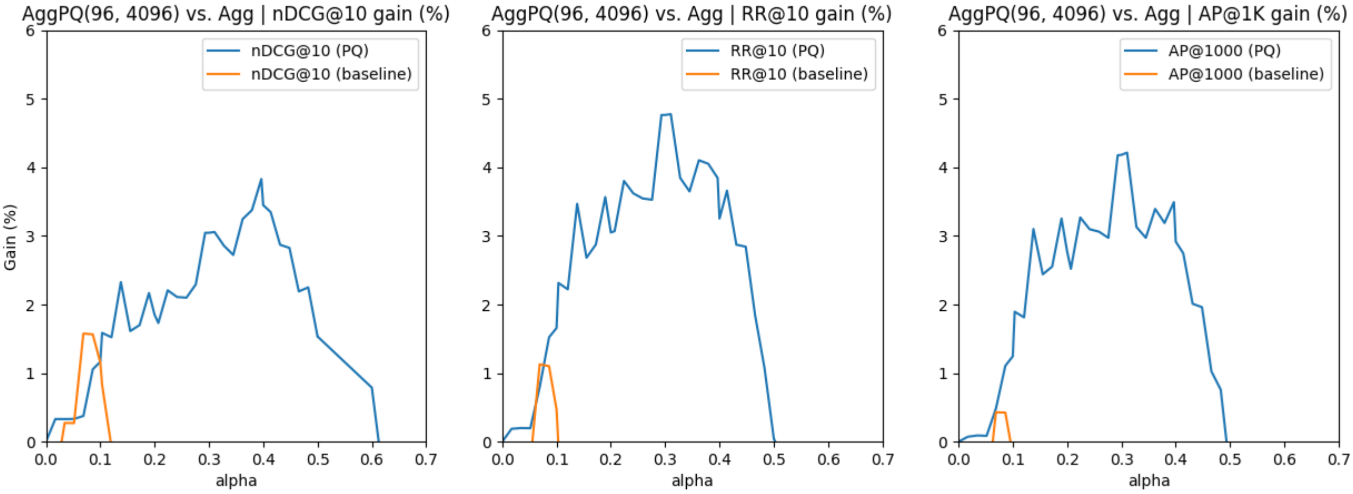


Figure A.8: Performance gain comparison between Aggretriever and Aggretriever-PQ, with BM25 interpolated scores, at various values of alpha, measured on MSMARCO Passage Dev

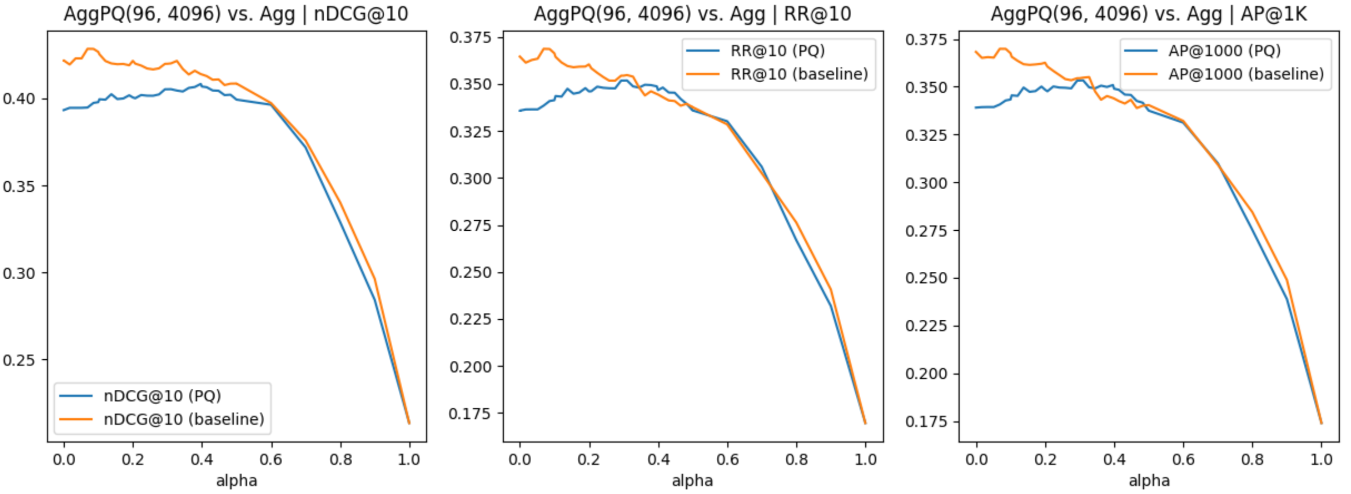


Figure A.9: Reranking performance comparison between Aggretriever and Aggretriever-PQ, with BM25 interpolated scores, at various values of alpha, measured on MSMARCO Passage Dev

Model	M	K	Index Size (MB)	Alpha	RR@10	Interp. Gain
BM25	-	-	-	-	0.170	-
BM25 + TCT-ColBERT-PQ	32	512	339	0	0.187	-
BM25 + TCT-ColBERT-PQ	16	256	170	0	0.061	-
BM25 + TCT-ColBERT-PQ-I	16	256	170	0.1	0.198	+224.6%
BM25 + TCT-ColBERT-PQ	48	512	491	0	0.228	-
BM25 + TCT-ColBERT-PQ	24	1024	290	0	0.142	-
BM25 + TCT-ColBERT-PQ-I	24	1024	290	0.1	0.229	+61.3%
BM25 + TCT-ColBERT-PQ	96	512	946	0	0.278	-
BM25 + TCT-ColBERT-PQ	64	1024	712	0	0.252	-
BM25 + TCT-ColBERT-PQ-I	64	1024	712	0.1	0.278	+10.32%
BM25 + TCT-ColBERT	-	-	27852.8	0	0.359	
BM25 + TCT-ColBERT-I	-	-	27852.8	0.1	0.357	-0.55%

Table A.4: Relationship between alpha and index size for TCT-ColBERT on MSMARCO Passage Dev

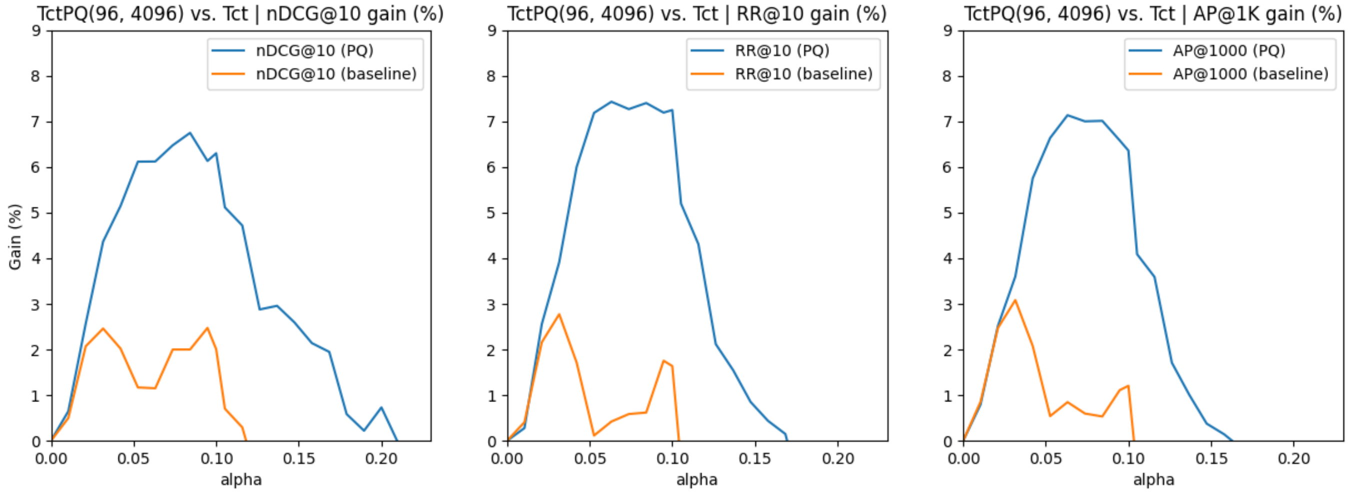


Figure A.10: Performance gain comparison between TCT-ColBERT and TCT-ColBERT-PQ, with BM25 interpolated scores, at various values of alpha, measured on MSMARCO Passage Dev

Model	M	K	Index Size (MB)	Alpha	RR@10	Interp. Gain
BM25	-	-	-	-	0.170	-
BM25 + Aggretriever-PQ	8	2048	133	0	0.210	-
BM25 + Aggretriever-PQ	8	256	102	0	0.151	-
BM25 + Aggretriever-PQ-I	8	256	102	0.3	0.218	+44.4%
BM25 + Aggretriever-PQ	24	1024	290	0	0.267	-
BM25 + Aggretriever-PQ	16	2048	225	0	0.259	-
BM25 + Aggretriever-PQ-I	16	2048	225	0.3	0.268	+3.5%
BM25 + Aggretriever	-	-	27852.8	0	0.368	-
BM25 + Aggretriever-I	-	-	27852.8	0.3	0.354	-3.8%

Table A.5: Relationship between alpha and index size for Aggretriever on MSMARCO Passage Dev

Model	M	K	Index Size (GB)	Alpha	Compression	NDCG@10	Drop
BM25	-	-	-		-	0.497	-
BM25 + ColBERT	-	-	143*		-	0.701*	-
BM25 + TCT-ColBERT	-	-	27.2		-	0.699	-
BM25 + Aggretriever	-	-	27.2		-	0.707	-
BM25 + TCT-ColBERT-I	-	-	27.2		-	0.719	-
BM25 + Aggretriever-I	-	-	27.2		-	0.703	-
BM25 + ColBERT-PQ	16	256	10.2*		x14.01	0.684*	-2.3%
BM25 + TCT-ColBERT-PQ-I	96	2048	1.13		x24.07	0.685	-4.72%
BM25 + TCT-ColBERT-PQ-I	96	4096	1.23		x22.11	0.680	-5.42%
BM25 + Aggretriever-PQ-I	96	4096	1.23		x22.11	0.698	-0.71%

Table A.6: Reranking performence on MSMARCO Passage TREC DL19

Model	M	K	Index Size (GB)	Alpha	Compression	NDCG@10	Drop
BM25	-	-	-		-	0.488	-
BM25 + ColBERT	-	-	143*		-	0.723*	-
BM25 + TCT-ColBERT	-	-	27.2		-	0.680	-
BM25 + Aggretriever	-	-	27.2		-	0.704	-
BM25 + TCT-ColBERT-I	-	-	27.2		-	0.693	-
BM25 + Aggretriever-I	-	-	27.2		-	0.717	-
BM25 + ColBERT-PQ	16	256	10.2*		x14.01	0.714*	-2.3%
BM25 + TCT-ColBERT-PQ-I	96	2048	1.13		x24.07	0.667	-3.75%
BM25 + TCT-ColBERT-PQ-I	96	4096	1.23		x22.11	0.672	-3.0%
BM25 + Aggretriever-PQ-I	96	4096	1.23		x22.11	0.705	-1.67%

Table A.7: Reranking performence on MSMARCO Passage TREC DL 2020

A.3. Supervized quantization for re-ranking

A.3.1. Pre-training

Model (pre-trained only)	M	K	Index Size (GB)	RR@10
BM25	-	-	-	0.170
BM25 + TCT-ColBERT-PQ	16	256	0.166	0.061
BM25 + TCT-ColBERT-PQ	24	1024	0.283	0.142
BM25 + TCT-ColBERT-PQ	96	256	0.824	0.284
BM25 + TCT-ColBERT-WolfPQPre	16	256	0.166	0.053
BM25 + TCT-ColBERT-WolfPQPre	24	1024	0.283	0.123
BM25 + TCT-ColBERT-WolfPQPre	96	256	0.824	0.310

Table A.8: Reranking performence on MSMARCO Passage Dev

A.3.2. Listwise loss vs. Pointwise loss

Model	M	K	Index Size (GB)	Alpha	LIP	#Epochs	RR@10	Drop
BM25	-	-	-	-	-	-	0.170	-
BM25 + TCT-ColBERT	-	-	27.2	0	-	-	0.359	-
BM25 + TCT-ColBERT-l	-	-	27.2	0.1	-	-		-
BM25 + TCT-ColBERT-PQ	16	256		0.0	-	-	0.061	
BM25 + TCT-ColBERT-PQ-l	16	256		0.1	-	-	0.198	
BM25 + TCT-ColBERT-WolfPQ	16	256	0.166	0	0	25	0.168	
BM25 + TCT-ColBERT-WolfPQ	16	256	0.166	0	0.3	25	0.159	
BM25 + TCT-ColBERT-WolfPQ	16	256	0.166	0	0.5	25	0.168	
BM25 + TCT-ColBERT-WolfPQ	16	256	0.166	0	0.7	30	0.161	
BM25 + TCT-ColBERT-WolfPQ	16	256	0.166	0	1.0	30	0.140	
BM25 + TCT-ColBERT-WolfPQ-l	16	256	0.166	0.1	0	25	0.225	
BM25 + TCT-ColBERT-WolfPQ-l	16	256	0.166	0.1	0.3	25	0.220	
BM25 + TCT-ColBERT-WolfPQ-l	16	256	0.166	0.1	0.5	25	0.215	
BM25 + TCT-ColBERT-WolfPQ-l	16	256	0.166	0.1	0.7		0.219	
BM25 + TCT-ColBERT-WolfPQ-l	16	256	0.166	0.1	1.0	30	0.186	
BM25 + TCT-ColBERT-WolfPQl-l	16	256	0.166	0.1	0	25	0.225	
BM25 + TCT-ColBERT-WolfPQl-l	16	256	0.166	0.1	0.01	25	0.212	
BM25 + TCT-ColBERT-WolfPQl-l	16	256	0.166	0.1	0.05	30	0.211	
BM25 + TCT-ColBERT-WolfPQl-l	16	256	0.166	0.1	0.1	25	0.229	
BM25 + TCT-ColBERT-WolfPQl-l	16	256	0.166	0.1	0.3	30	0.235	
BM25 + TCT-ColBERT-WolfPQl-l	16	256	0.166	0.1	0.5	30	0.236	
BM25 + TCT-ColBERT-WolfPQl-l	16	256	0.166	0.1	0.7	30	**0.242**	
BM25 + TCT-ColBERT-WolfPQl-l	16	256	0.166	0.1	1.0	30	0.212	

Table A.9: Reranking performence on MSMARCO Passage Dev

A.3.3. Exploration of semantic sampling

Model	M	K	Index Size (GB)	Alpha	MDF	#Epochs	RR@10
BM25 + TCT-ColBERT-WolfPQ-l	16	256	0.166	0.1	-	25	0.225
BM25 + TCT-ColBERT-WolfPQSem-l	16	256	0.166	0.1	5.0	40	0.129
BM25 + TCT-ColBERT-WolfPQSem-l	16	256	0.166	0.1	25.0	25	0.222
BM25 + TCT-ColBERT-WolfPQSem-l	16	256	0.166	0.1	30.0	30	**0.258**
BM25 + TCT-ColBERT-WolfPQSem-l	16	256	0.166	0.1	35.0	25	0.241
BM25 + TCT-ColBERT-WolfPQSem-l	16	256	0.166	0.1	55.0	30	0.237

Table A.10: Reranking performence on MSMARCO Passage Dev

Model	**M**	**K**	**Index Size (GB)**	**Alpha**	**MDF**	**RR@10**
BM25 + TCT-ColBERT-WolfPQ	16	256	0.166	-	-	0.168
BM25 + TCT-ColBERT-WolfPQSem	16	256	0.166	-	30.0	**0.172**
BM25 + TCT-ColBERT-WolfPQ-I	16	256	0.166	0.1	-	0.225
BM25 + TCT-ColBERT-WolfPQSem-I	16	256	0.166	0.1	30.0	**0.258**

Table A.11: Reranking performence on MSMARCO Passage Dev

A.3.4. Comparison to other re-ranking models

Model	**M**	**K**	**Index Size (GB)**	**NDCG@10 (2019)**	**NDCG@10 (2020)**
BM25	-	-	-	0.497	0.488
TCT-ColBERT	-	-	27.2	0.670*	-
JPQ	-	-	0.83*	0.677*	-
RepCONC	-	-	0.393*	0.668*	-
BM25 + TCT-ColBERT	-	-	27.2	0.699	0.680
BM25 + TCT-ColBERT-I	-	-	27.2	0.719	0.693
BM25 + ColBERT-CQ	16	256	10.2*	0.704*	0.716*
TCT-ColBERT-PQ	16	256	0.166	0.085	0.046
TCT-ColBERT-PQ	24	1024	0.283	0.268	0.169
TCT-ColBERT-PQ	96	256	0.824	0.608	0.544
BM25 + TCT-ColBERT-PQ	16	256	0.166	0.203	0.220
BM25 + TCT-ColBERT-PQ	24	1024	0.283	0.424	0.339
BM25 + TCT-ColBERT-PQ	64	1024	0.694	0.623	0.597
BM25 + TCT-ColBERT-PQ	96	256	0.824	0.636	0.604
BM25 + TCT-ColBERT-WolfPQ	16	256	0.166	0.380	0.387
BM25 + TCT-ColBERT-WolfPQ	24	1024	0.283	0.522	0.537
BM25 + TCT-ColBERT-WolfPQ	96	256	0.824	0.668	0.627
BM25 + TCT-ColBERT-WolfPQSem	16	256	0.166	0.402	0.373
BM25 + TCT-ColBERT-WolfPQSem	24	1024	0.283	0.541	0.528
BM25 + TCT-ColBERT-WolfPQSem	96	256	0.824	0.656	0.614
BM25 + TCT-ColBERT-PQ-I	16	256	0.166	0.549	0.558
BM25 + TCT-ColBERT-PQ-I	24	1024	0.283	0.603	0.548
BM25 + TCT-ColBERT-PQ-I	64	1024	0.694	0.679	0.659
BM25 + TCT-ColBERT-PQ-I	96	256	0.824	0.685	0.658
BM25 + TCT-ColBERT-WolfPQ-I	16	256	0.166	0.521	0.535
BM25 + TCT-ColBERT-WolfPQ-I	24	1024	0.283	0.608	0.590
BM25 + TCT-ColBERT-WolfPQ-I	96	256	0.824	0.696	0.660
BM25 + TCT-ColBERT-WolfPQSem-I	16	256	0.166	0.532	0.519
BM25 + TCT-ColBERT-WolfPQSem-I	24	1024	0.283	0.626	0.612
BM25 + TCT-ColBERT-WolfPQSem-I	96	256	0.824	0.674	0.659
BM25 + TCT-ColBERT-WolfPQI-I	16	256	0.166	0.552	0.511
BM25 + TCT-ColBERT-WolfPQI-I	24	1024	0.283	0.6	0.58
BM25 + TCT-ColBERT-WolfPQI-I	96	256	0.824	**0.697**	**0.673**
BM25 + TCT-ColBERT-WolfPQSemI-I	16	256	0.166	0.519	0.485
BM25 + TCT-ColBERT-WolfPQSemI-I	24	1024	0.283	0.637	0.591
BM25 + TCT-ColBERT-WolfPQSemI-I	96	256	0.824	0.666	0.669

Table A.12: Reranking performence on MSMARCO Passage TREC DL 2019 and 2020 (the WolfPQ models are only trained on a small sample)

Model	M	K	Index Size (GB)	RR@10 (2019)	RR@10 (2020)
BM25	-	-	-	0.682	0.655
BM25 + TCT-ColBERT	-	-	27.2	0.840	0.815
BM25 + TCT-ColBERT-I	-	-	27.2	0.860	0.815
TCT-ColBERT-PQ	16	256	0.166	0.136	0.107
TCT-ColBERT-PQ	24	1024	0.283	0.485	0.326
TCT-ColBERT-PQ	96	256	0.824	0.8	0.731
BM25 + TCT-ColBERT-PQ	16	256	0.166	0.295	0.388
BM25 + TCT-ColBERT-PQ	24	1024	0.283	0.616	0.544
BM25 + TCT-ColBERT-PQ	96	256	0.824	0.793	0.794
BM25 + TCT-ColBERT-WolfPQ	16	256	0.166	0.596	0.538
BM25 + TCT-ColBERT-WolfPQ	24	1024	0.283	0.708	0.745
BM25 + TCT-ColBERT-WolfPQ	96	256	0.824	0.8	0.782
BM25 + TCT-ColBERT-WolfPQSem	16	256	0.166	0.598	0.558
BM25 + TCT-ColBERT-WolfPQSem	24	1024	0.283	0.728	0.713
BM25 + TCT-ColBERT-WolfPQSem	96	256	0.824	0.853	0.778
BM25 + TCT-ColBERT-PQ-I	16	256	0.166	0.762	0.707
BM25 + TCT-ColBERT-PQ-I	24	1024	0.283	0.799	0.727
BM25 + TCT-ColBERT-PQ-I	96	256	0.824	**0.864**	0.813
BM25 + TCT-ColBERT-WolfPQ-I	16	256	0.166	0.677	0.681
BM25 + TCT-ColBERT-WolfPQ-I	24	1024	0.283	0.757	0.721
BM25 + TCT-ColBERT-WolfPQ-I	96	256	0.824	0.81	0.772
BM25 + TCT-ColBERT-WolfPQSem-I	16	256	0.166	0.768	0.716
BM25 + TCT-ColBERT-WolfPQSem-I	24	1024	0.283	0.8	0.766
BM25 + TCT-ColBERT-WolfPQSem-I	96	256	0.824	0.832	0.82
BM25 + TCT-ColBERT-WolfPQI-I	16	256	0.166	0.767	0.697
BM25 + TCT-ColBERT-WolfPQI-I	24	1024	0.283	0.75	0.722
BM25 + TCT-ColBERT-WolfPQI-I	96	256	0.824	0.838	**0.824**
BM25 + TCT-ColBERT-WolfPQSemI-I	16	256	0.166	0.719	0.688
BM25 + TCT-ColBERT-WolfPQSemI-I	24	1024	0.283	0.847	0.788
BM25 + TCT-ColBERT-WolfPQSemI-I	96	256	0.824	0.816	0.783

Table A.13: Reranking performence on MSMARCO Passage TREC DL 2019 and 2020 (the WolfPQ models are only trained on a small sample)