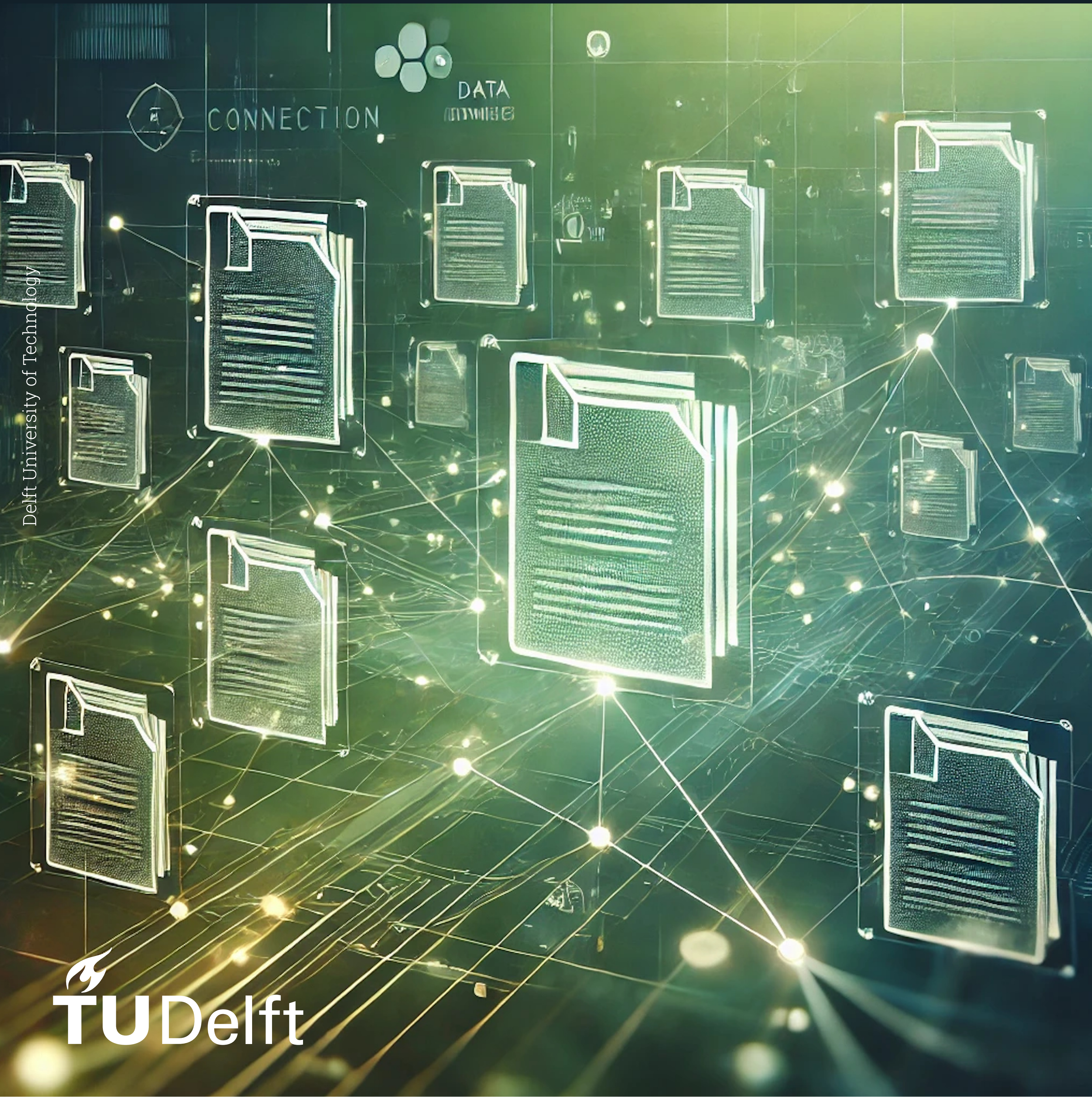
Resource Efficient Adaptive Retrieval

Martijn Smits



Delft University of Technology

TU Delft

Resource Efficient Adaptive Retrieval

by

Martijn Smits

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday May 7, 2025 at 14:00 PM.

Student number:	4569954	
Project duration:	September 30, 2024 – May 7, 2025	
Thesis committee:	Prof. Dr. Avishek Anand	Thesis Advisor
	Dr. Jurek Leonhardt	Daily Supervisor
	Dr. Julián Urbano,	Committee Member

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

TU Delft

Abstract

Adaptive retrieval is a technique to overcome the recall limitations of two-stage retrieval pipelines. Adaptive retrieval focuses mainly on effectiveness, but shows potential to improve efficiency. This research focuses on the trade-off between effectiveness and efficiency in adaptive retrieval. We explore the behaviour of neighbourhoods formed by the corpus graph, find that the effectiveness of adaptive retrieval varies across queries, and identify the most effective way to leverage the corpus graph. We investigate the impact of two different scoring mechanisms on the efficiency and effectiveness of an adaptive retrieval pipeline. We observe that score interpolation tends to improve adaptive retrieval's effectiveness, and incorporating an additional cross-encoder stage can lead to further gains. We compare our proposed solutions against several baselines to examine the trade-offs between effectiveness and efficiency. In our setting, we find that adaptive retrieval can improve efficiency at the cost of effectiveness. For our experiments, we introduce a visualisation tool for graph exploration and an adaptive retrieval component for efficient retrieval pipelines.

Preface

I chose to study Computer Science and Engineering for my BSc because I am interested in problem-solving. Computer Science gives me the tools to solve problems and bring solutions to life. Enjoying the subject and its challenges, I decided to continue my studies with an MSc in Computer Science and Engineering to deepen my knowledge and develop my skills. This interest eventually led me to Information Retrieval, which immediately captured my attention due to its imperfect and subjective nature, offering endless opportunities for improvement. With the rapid growth of digital information, organising and retrieving data has become essential to making sense of an increasingly complex digital landscape.

While the courses in the program provide a strong foundation across various areas of the field, the Master's thesis allows for an in-depth exploration of a specific topic. It demonstrates that even when you initially think you have a strong understanding of a topic, you later realise there's much more to it than you first thought. It was both a challenging and enjoyable experience to dive so deeply into a topic. The topic of adaptive retrieval was highly engaging to explore, as it was a relatively new concept with great potential. This area has much undiscovered potential, and it was exciting to contribute to its development.

The master's thesis is a challenge unlike anything else in the BSc and MSc programs, and I could not have done it without my supervisors. I want to thank Jurek Leonhardt, my daily supervisor, for his invaluable support throughout this project. I sincerely appreciate the time he dedicated to our weekly meetings and his help in addressing various questions during my work. His insights into the problem were invaluable in shaping my approach, and his overall assistance played a significant role in completing this thesis. Moreover, I want to thank Avishek Anand, my thesis advisor, for his guidance and support throughout this project. His role in shaping the direction of my research was invaluable, and his perspective provided crucial insights that helped refine my approach. His feedback was instrumental in improving the quality of my work, and I am grateful for his involvement and expertise. I would also like to express my appreciation for the research group. I particularly enjoyed being part of a small research group, which allowed for more personalised attention and closer collaboration. The regular talks and presentations were insightful and helped strengthen the sense of community within the group, making the experience even more enjoyable and collaborative. Finally, I want to thank my family and friends for supporting me throughout this journey.

Martijn Smits
Delft, April 2025

Contents

List of Figures

List of Tables

1 Introduction

Ad-hoc retrieval refers to retrieving relevant documents from an extensive, unstructured collection in response to a one-time user query. These queries are typically expressed in natural language and aim to satisfy a specific information need [39]. Ad-hoc retrieval presents several challenges that make it a complex problem in information retrieval. Queries are frequently underspecified and phrased differently from the content itself. Additionally, relevance is subjective and can differ significantly from one user to another. On top of that, the unstructured nature of document corpora adds complexity, as there are no predefined relationships between the data, making it difficult for retrieval systems to grasp the underlying meaning of text.

Traditionally, ad-hoc retrieval relied on basic lexical methods [37, 36], where the overlap of words between documents and queries primarily determined relevance. However, in recent years, the rise of neural models [43, 7, 28, 33] has significantly changed the field. These models improve semantic understanding and capture contextual relationships in natural language tasks. They have enabled the development of neural retrieval models [27, 30, 21, 41, 31, 32], which perform much better at capturing query intent than lexical models. Neural ranking models require substantial time and resources to compute document relevance scores. A common solution to this issue is the two-stage retrieval pipeline. Instead of ranking every document with a neural model, a lexical retriever is first used to gather candidate documents. Then, relevance scores are computed for this subset using a re-ranking model. A common approach for re-ranking is the cross-encoder architecture, which means that documents and queries are processed as pairs into relevance scores.

Despite the effectiveness of this approach, the two-stage pipeline suffers from the bounded recall problem [22] (see Figure 1.1). The initial retriever is highly efficient but lacks effectiveness, regularly failing to retrieve all relevant documents within its top-ranked results. Since the re-ranking stage can only score documents returned by the first stage, any relevant documents missed in the initial retrieval are never considered. As a result, the overall recall of the pipeline is inherently limited by the performance of the first-stage retriever.

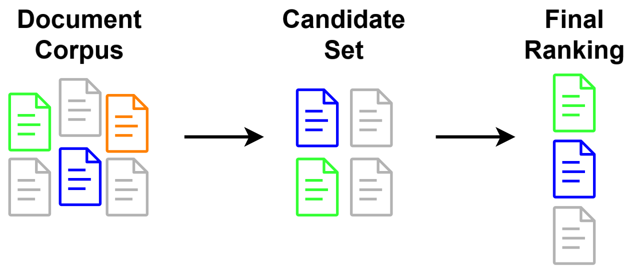


Figure 1.1: The bounded recall problem. The initial retriever misses the relevant orange document. Therefore, it is not in the candidate set, and the re-ranker never sees the document.

Adaptive retrieval [22, 17, 34, 35] is designed to address the bounded recall problem. Adaptive retrieval considers additional documents during the re-ranking stage of the pipeline. It builds on the clustering hypothesis, suggesting that closely related documents are likely relevant to the same queries [22]. Adaptive retrieval leverages this hypothesis by expanding the candidate set with documents related to highly ranked documents during the re-ranking stage. This allows for documents that the initial retriever missed to be found. Closely related neighbour documents are often stored in a Corpus Graph [22], which models documents as nodes with edges to other documents based on their similarity. Corpus Graphs can be pre-computed and allow for simple look-ups during query processing. It is an efficient method to consider neighbours at the cost of additional memory usage to store the graph. The main objective of adaptive retrieval is to increase the recall of the candidate set. Figure 1.2 illustrates how adaptive retrieval can help find relevant documents that may have been initially overlooked.

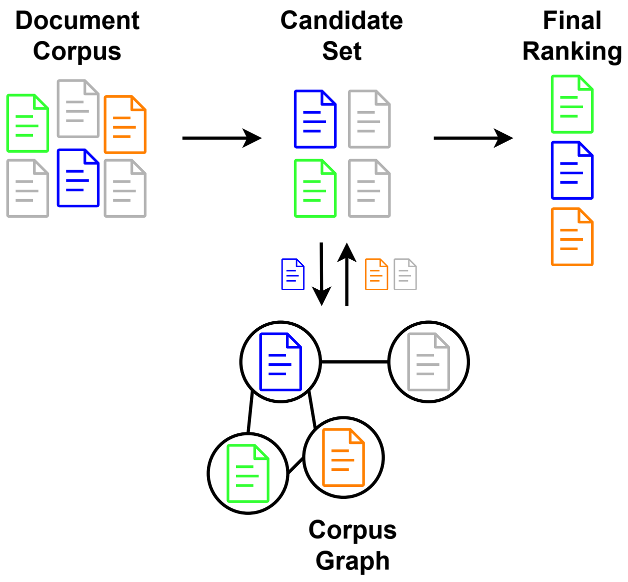


Figure 1.2: The general idea of adaptive retrieval. The initial retriever misses the relevant orange document and is initially not included in the candidate set. Its relation to the blue and green document in the corpus graph allows it to be found during retrieval.

While effectiveness is key in information retrieval, efficiency can be equally important depending on the demands of real-world applications. Efficiency is typically characterised by two main factors: latency and resource usage. Low latency is essential in applications where users expect fast responses and systems are required to handle large volumes of data. Low latency is one of the key factors in the user experience of retrieval systems. Resource usage is another important aspect of retrieval, as it directly influences retrieval systems' cost and energy consumption. In an era where environmental sustainability is a pressing concern and neural models consume increasingly more power, efficient resource allocation has become more important than ever.

Cross-encoder-based retrieval systems focus on high effectiveness, while often not being very efficient. Dense retrieval [13, 44, 19] offers a more efficient approach by processing the queries and documents separately into a common vector space and computing similarity scores between the vectors to produce a similarity score. This approach means that documents can be pre-computed and only the query needs to be embedded during query processing. Dense retrieval improvements in efficiency are mainly in terms of latency, as nearest neighbour search often still relies on GPUs to perform well. More efficient retrieval systems [18, 45, 46] use the same principles as dense retrieval by processing queries and documents separately. However, a more common approach is to use these vectors in a re-ranking setting, where documents from the initial retriever are re-ranked based on the similarity scores. This eliminates the need for nearest neighbour search, often allowing such solutions to perform well on CPUs.

Optimising retrieval systems involves balancing efficiency and effectiveness, as improvements in one often come at the expense of the other. In this work, we carefully study this balance in the context of adaptive retrieval. Originally, adaptive retrieval was designed to improve the effectiveness of retrieval systems. However, its reliance on efficient graph look-ups makes it a promising approach for improving effectiveness within efficient retrieval settings. Our work builds further on `LADR` [17], an efficient adaptive retrieval solution, by continuing research on the trade-off between efficiency and effectiveness.

In this research, we aim to answer the following questions:

RQ1: How can a corpus graph be used most effectively to overcome the limitations of first-stage lexical retrievers?

RQ2: How do scoring mechanisms influence the effectiveness of adaptive retrieval?

RQ3: How does adaptive retrieval affect efficient re-ranking in effectiveness and efficiency?

In our work, we investigate how corpus graphs can be leveraged to overcome the limitations of first-stage retrievers. While prior research has demonstrated the effectiveness of adaptive retrieval through corpus graphs [22, 17, 34], our focus is on deepening the understanding of the structure and behaviour of the graph itself. For our initial exploration, we introduce a visualisation tool called `GarVis`, designed to intuitively explore the neighbourhood structures and patterns within the corpus graph by analysing neighbourhoods across various queries. Using `GarVis`, we find that the effectiveness of adaptive retrieval varies by query, sometimes improving and other times diminishing performance. Moreover, we see patterns in document neighbourhoods that support the clustering hypothesis. Based on these insights, we systematically analyse neighbourhood structures to determine optimal budget configurations. Additionally, we examine the presence of duplicate documents within neighbourhoods and observe that, in most cases, these neighbourhoods contain many duplicates. To address this issue, we propose a strategy that removes duplicate documents from the candidate set.

Next, we investigate the effect of scoring mechanisms on adaptive retrieval. While adaptive retrieval aims to increase effectiveness by increasing the recall of the candidate set, it does not directly affect the scores. A popular technique in efficient retrieval is score interpolation, which combines the lexical and semantic scores instead of discarding the lexical score. It is a relatively inexpensive operation that has shown notable improvements in effectiveness [18]. We investigate the effect of interpolation-based re-ranking and find that interpolation positively affects the effectiveness in an adaptive retrieval setting. Moreover, we experiment with cross-encoders to improve our ranking further. We extend our pipeline with an additional stage at the end, where we re-rank a small portion of our results using a cross-encoder. We find that with a small cross-encoder budget, we can gain notable improvements in effectiveness.

Finally, we explore the efficiency and effectiveness trade-offs between our solution and several baselines. We find that adding adaptive retrieval into an efficient retrieval pipeline generally improves the efficiency at the cost of some of its effectiveness. This result is surprising, as adaptive retrieval improves effectiveness while introducing additional overhead. However, it can be attributed to the deduplication of the candidate set, which lowers the re-ranking cost but reduces effectiveness. Furthermore, we conduct a more detailed comparison of efficiency in cross-encoder-based solutions. We find efficient adaptive retrieval pipelines perform well under a small cross-encoder budget, whereas effective adaptive retrieval excels with larger budgets.

The structure of this work is as follows: we begin in Chapter 2 with background information and an overview of related work in the field. Chapter 3 elaborates on the problem setting and introduces the methods we use to address our research questions. In Chapter 4, we describe our experimental setup, including the datasets used, evaluation metrics, baseline systems for comparison, and other relevant implementation details. Chapter 5 presents and discusses the results of our experiments. Finally, we conclude in Chapter 6, where we summarise our findings and suggest directions for future research.

2 Background and Related Work

2.1. Ad-hoc Retrieval

Ad-hoc retrieval is retrieving relevant documents from an unstructured collection in response to a one-time user query, typically expressed in natural language, to address a specific information need [39]. This process involves searching through a corpus of unstructured documents and ranking them based on their relevance to the query. The primary goal is to ensure that the most relevant documents appear at the top of the results (see Figure 2.1).

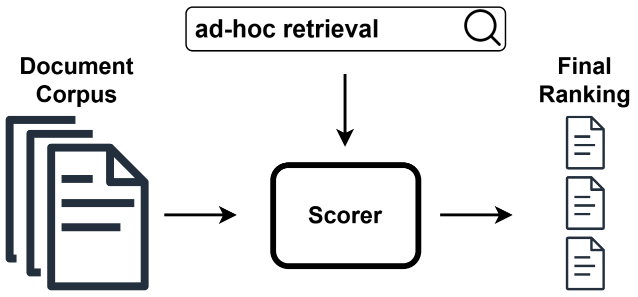


Figure 2.1: The general idea of ad-hoc retrieval. A scorer ranks documents from a corpus based on their estimated relevance to a given query.

2.1.1. Two-stage Retrieval

The two-stage retrieval pipeline (Figure 2.2) is a widely employed approach in information retrieval systems. It first uses a simple retrieval method to generate an initial candidate set of documents, followed by a more computationally intensive re-ranking model that effectively scores a subset of documents from the candidate set.

Initial Retrievers

Retrieval systems are most often used to search through large corpora of documents, making it infeasible to use computationally expensive ranking models to evaluate the relevance of each document to the given query. The purpose of the first-stage retriever is to efficiently narrow down a pool of potentially relevant documents. For a given query, the first-stage retriever quickly evaluates the relevance of each document using an inexpensive scoring function, producing an initial ranking that is passed on to the next stage.

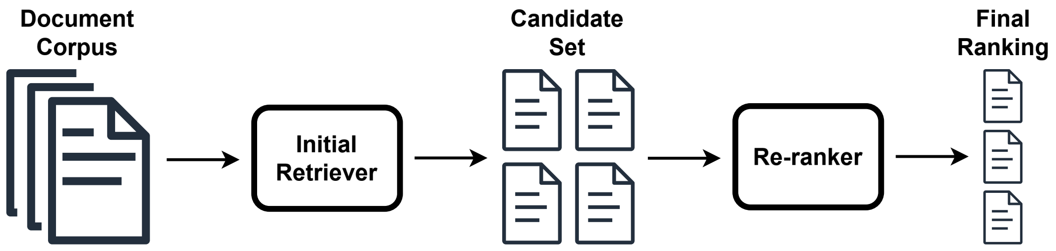


Figure 2.2: The Two-stage Retrieval Pipeline. An initial retriever produces a candidate set of documents, which is then re-ranked by a more efficient re-ranker.

A frequently used initial retrieval method is lexical retrieval, which searches for documents based on exact term matches in a query. The most common techniques involve storing all terms in an index, which points to the documents where those terms appear. These indices are pre-computed and can be quickly accessed during query processing for efficient lookups. However, simple lookups in the index are not enough to retrieve documents from the corpus effectively. `BM25` [37, 36] is a more sophisticated method of lexical retrieval. `BM25` still relies on the same basic concept of matching terms between the query and the document. However, it introduces a scoring mechanism that accounts for the frequency of query terms in the document while adjusting for the rarity of those terms across the entire corpus. This means that terms found in a few documents are considered more important than terms found in many documents.

Given a document D and a query Q, the `BM25` score is computed as follows [40]:

$$\text{BM25}(D,Q) = \sum_{i=1}^{n} \ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D) + k_1\cdot(1 - b + b\cdot\frac{|D|}{\text{avgdl}})} \tag{2.1}$$

This equation sums up the contribution from each query term $\{q_i, ..., q_n\} \in Q$ appearing in the query. The first part of the equation is the inverse document frequency (IDF) component that gives higher weight to rarer terms, where N is the total number of documents in the collection and $n(q_i)$ is the number of documents containing the term q_i. The second part of the equation adjusts for term frequency within document D, where $f(q_i, D)$ represents the frequency of term q_i in document D. The fraction $\frac{|D|}{\text{avgdl}}$ compares the document length $|D|$ to the average document length $avgdl$ in the document corpus, ensuring that longer documents do not unfairly dominate the ranking. Parameter k_1 controls how much the term frequency influences the score, while b controls the normalisation of document length. Under normal circumstances, these parameters are often chosen as $k_1 = 1.2$ and $b = 0.75$.

While `BM25` has been a strong baseline for lexical retrieval, it relies on exact term matching and struggles to capture semantic relationships between words. Learned sparse retrieval [11, 4, 3, 26, 9] addresses this issue by learning sparse representations for documents and queries. These representations enable efficient first-stage retrieval using exact term matching and inverted indexes, preserving the desirable properties of traditional lexical approaches.

Re-rankers

Initial retrievers are highly efficient at identifying suitable sets of candidate documents, but often lack effectiveness [42]. The second stage improves effectiveness by employing re-ranking models that reorder the initial ranking of documents by leveraging semantic features and aligning results more closely with the query's intent.

Pre-trained transformer models have revolutionised document ranking, improving relevance modelling and retrieval effectiveness. The transformer architecture [43] is based on self-attention mechanisms that enable models to capture long-range dependencies in text sequences. This allows transformers to learn contextual relationships between words, understand grammar structures, and adapt to nearly any language-related task, making them highly versatile for many applications, including re-ranking. The transformer architecture allows for high parallelisation and can reach state-of-the-art performance with comparatively short training times [43]. This efficiency allows for more extensive training datasets, significantly improving model performance. Several well-known transformer models include BERT [7], GPT [28] and T5 [33].

A common approach to employing the transformer architecture in re-ranking is through cross-encoders (see Figure 2.3). These models process document-query pairs as a single input and produce a relevance score for the pair while capturing intricate query-document relations, improving performance at higher computational costs. Cross-encoder-based re-rankers can generally be categorised into three main approaches: pointwise, pairwise and listwise ranking models. Pointwise models, such as monoBERT [27], monoT5 [30], and RankLLaMa [21], treat re-ranking as a sequence generation task, where the model estimates relevance scores for individual query-document pairs. Pairwise models, like duoBert [27] and duoT5 [30], compare two candidate documents simultaneously to determine their relative ranking, accounting for relative preference between documents. Listwise models, such as RankGPT [41], RankVicuna [31] and RankZephyr [32], consider multiple documents simultaneously, aiming to optimise the overall ranking order immediately.

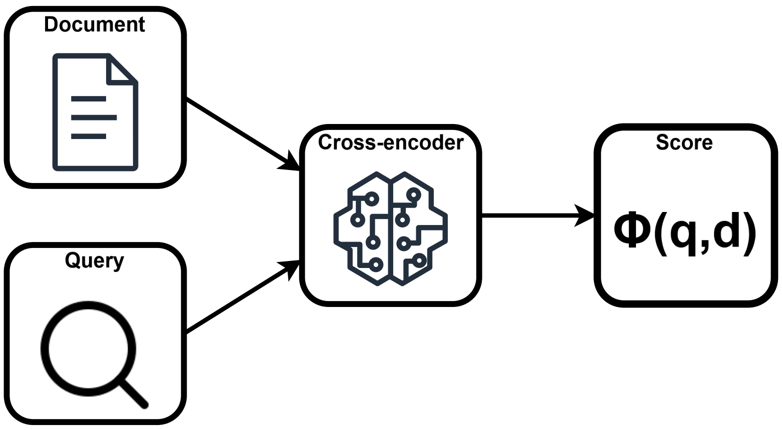


Figure 2.3: Cross-encoder based re-ranking. Documents and queries are input into a cross-encoder, which evaluates the document's relevance to the query.

2.1.2. Dense Retrieval

The cross-encoder architecture is highly effective for capturing query-document relations; however, it requires significant computational resources to process the query and the document for each candidate pair. Dense retrieval uses the dual-encoder architecture (see Figure 2.4) that independently embeds queries and documents into a common vector space. These vector representations encode the underlying meaning of the documents and queries, meaning we can directly compare the semantic similarity between documents and queries by comparing these vectors. This allows for pre-computation of document embeddings, meaning only the query needs to be encoded during query processing time.

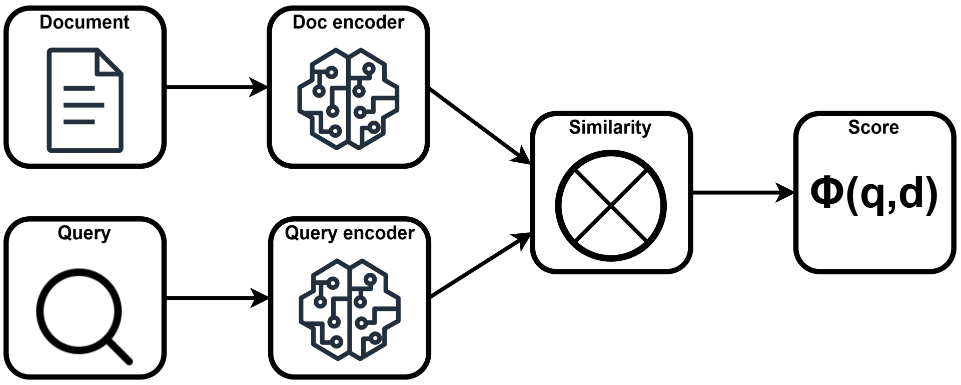


Figure 2.4: Dual-encoder based re-ranking. Documents and queries are encoded separately by an encoder, and the similarity between their vectors is computed to generate a relevance score.

Search Techniques

The query is encoded and compared to the document vectors during processing. One straightforward approach for this comparison is *exact matching*, which directly computes the similarity between the query vector and each document vector. This method ensures that all nearest neighbours are found according to the chosen similarity metric. However, it is computationally intensive and slow when applied to large corpora. A more efficient approach is to use approximate nearest neighbour (ANN) search algorithms.

A widely used method of ANN employs Hierarchical Navigable Small World (HNSW) graphs [25, 29], which are structured as multiple layers of proximity graphs (see Figure 2.5). Each node in the graph represents a vector whose edges indicate the distance to another vector. The lowest layer includes all nodes, while higher layers progressively contain fewer nodes. The search process begins at the highest layer, where the local minimum distance is identified. From there, it moves to the next layer, continuing this process until the approximate nearest neighbours are found in the lowest layer.

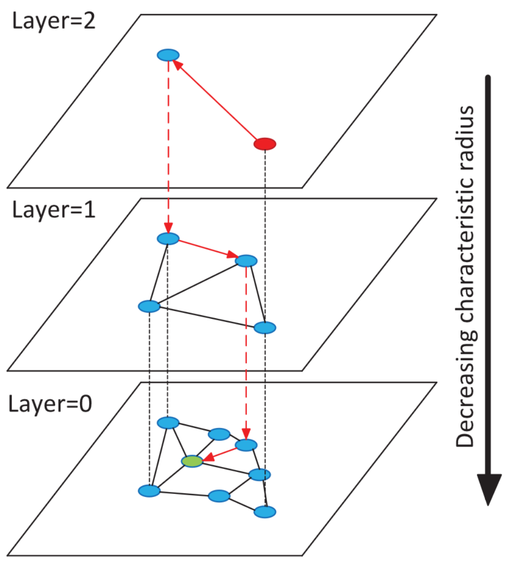


Figure 2.5: Illustration of the HNSW idea. The search starts from an element from the top layer (shown red). Red arrows show the direction of the greedy algorithm from the entry point to the query (shown green) [25].

Alternatively, another effective technique is combining Product Quantisation (PQ) with Inverted Files (IVF) [15, 16]. PQ compresses high-dimensional vectors by partitioning them into smaller sub-vectors, each replaced by the closest match from a predefined codebook. As a result, each vector is represented by a sequence of references to these codebook entries. IVF organises the dataset into clusters based on distance metrics, enabling faster searches by limiting the search space to only the relevant clusters.

Additionally, the Scalable Nearest Neighbours (ScaNN) index [12] provides another technique for ANN. ScaNN improves upon PQ by optimising for inner product preservation instead of minimising reconstruction error as in standard PQ. They propose a score-aware quantisation loss function, which gives more importance to accurately approximating vectors with higher true inner products. This leads to an anisotropic loss, meaning that errors parallel with the data point are more penalised than orthogonal errors. This means that the most significant features of the data are preserved, improving the accuracy of tasks like nearest neighbour search.

$$a \cdot b = \sum_{i=1}^{n} a_i b_i \tag{2.2}$$

Distance and similarity metrics are essential for comparing query and document vectors. Similarity metric measures how alike two vectors are, with higher values indicating higher similarity. The dot product (see Equation (2.2)) is a similarity metric that assesses the inner product of two vectors. Cosine similarity (see Equation (2.3)) calculates the cosine of the angle between the two values. Similarity metrics can be used to produce similarity scores between documents and vectors [10, 12]; by applying the $argmax$ operation, the documents with the highest similarity to the query can be identified. Distance metrics quantify the dissimilarity between two vectors, where smaller values indicate closer proximity. Euclidean distance is the most widely used distance metric, representing the exact spatial separation between vectors. Distance metrics are often used for distance calculations in clustering and graph-based ANN techniques [10, 15, 25].

$$S_c(a, b) = \frac{a \cdot b}{||a||||b||} \tag{2.3}$$

In this section, we have primarily focused on search algorithms over vector spaces, but dense representations are also effective in re-ranking scenarios [18]. In such settings, an initial retriever produces a set of candidate documents, which are then scored by computing the similarity between the query and each candidate document.

Embedding Models

Embedding models are a crucial component of dense retrieval systems. They transform textual data into dense vector representations that can be efficiently compared. These models learn to capture the semantic content of documents and queries, ensuring that similar items are mapped to vectors that are close together in the embedding space. Many of these models are based on transformer architecture because they capture long-range dependencies and contextual information.

Knowledge distillation is a widely used technique to enhance the efficiency of embedding models while minimising the loss in retrieval performance [19, 20, 13]. Knowledge distillation involves transferring knowledge from a large teacher to a smaller student model. The student model learns to approximate the teacher's effectiveness while being more computationally efficient. The student is trained using soft labels, the relevance scores predicted by the teacher model, alongside the hard labels, the actual relevance labels.

Contrastive learning is a popular method to improve the quality of the embedding space [44, 19, 20, 13] by bringing similar documents closer together while pushing dissimilar ones apart. The embedding models are trained using query-document pairs, where relevant documents are considered positive examples and irrelevant documents are treated as negative samples. Contrastive learning on all query-document pairs in the corpus is a slow process; therefore, most embedding models create a selection of hard negatives for training. ANCE [44] leverages ANN to sample negative examples to ensure that the negatives are not randomly chosen, but close enough to the query to learn subtle differences. Balanced Topic Aware Sampling (`TasB`) [13] improves sampling by introducing two strategies for creating batches. First, it clusters queries into topics to sample batches with stronger signals for in-batch negative learning. Next, it balances pairwise teacher score margins to ensure the model trains on examples of different difficulty.

2.1.3. Hybrid Ranking

In the two-stage pipeline, an initial retriever retrieves a candidate set, which is then re-ranked by a separate re-ranking model. The scores from the initial retriever are discarded, and the final ranking is determined solely by the scores from the re-ranker. In hybrid ranking, the results from the initial retriever are not discarded and are used alongside the re-ranker scores, complementing each other.

One hybrid approach is interpolation-based re-ranking [18], where the scores from the initial retriever are combined with the scores from the re-ranker through interpolation (see Equation (2.4)). In this equation, $\phi_S(q, d)$ represents the score from the initial retriever, and $\phi_D(q, d)$ represents the score from the re-ranker. The two scores are combined using the hyperparameter α. Since the scores are not normalised, α must be tuned for specific retrievers and datasets.

$$\phi(q, d) = \alpha \cdot \phi_S(q, d) + (1 - \alpha) \cdot \phi_D(q, d) \tag{2.4}$$

Another hybrid approach is reciprocal rank fusion [2], which combines the ranks from different retrievers (see Equation (2.5)). In this equation, $\pi_S(q,d)$ represents the rank from the initial retriever, and $\pi_D(q,d)$ represents the rank from the re-ranker. Additionally, η is a hyperparameter typically set to $\eta = 60$.

$$\phi(q,d) = \frac{1}{\eta + \pi_S(q,d)} + \frac{1}{\eta + \pi_D(q,d)} \tag{2.5}$$

2.2. Adaptive Retrieval

Despite its effectiveness in ranking documents, the two-stage retrieval pipeline is hindered by bounded recall. This means that if the initial retriever fails to identify certain documents, they are excluded from the candidate pool and, therefore, cannot be re-ranked during the second stage of the process. Adaptive retrieval is a technique to address this problem by considering additional documents during the re-ranking stage of the pipeline. Adaptive retrieval builds on the clustering hypothesis, which suggests that documents closely related to each other are likely relevant to the same queries [22]. That means the re-ranking pool can be expanded by introducing similar documents to the highest-scoring documents.

The core of adaptive retrieval is the corpus graph [22], which is a graph that connects similar documents. The nodes in a corpus graph represent all the documents in a document corpus D, while the edges are based on similarity scores between the documents. The similarity between documents is calculated similarly to the similarity between queries and documents. Any retrieval pipeline that produces an explicit similarity score $\phi(q,d)$ can compute a similarity score between documents $d_1 \in D$ and $d_2 \in D$ by computing $\phi(d_1,d_2)$. The corpus graph is created by computing the similarity between each document in the document corpus D. Storing a fully connected graph is infeasible, so typically, only the top k edges are kept.

Graph-based Adaptive re-ranking (GAR) [22] introduces the idea of adaptive retrieval. As shown in Algorithm 1, GAR begins by selecting a small batch of documents from the initial ranking R_0, scoring them, and adding them to the re-ranked pool R_1. The neighbours of this batch are then added to a graph frontier F. In the next iteration, GAR scores a small batch from F, adds them to R_1, and again expands the frontier with their neighbours. This process alternates between R_0 and F, continuing until the budget c is reached.

Algorithm 1: Graph-based Adaptive Re-Ranking [22]

```
Input: Initial ranking $R_0$, batch size $b$, budget $c$, corpus graph $G$
Output: Re-Ranked pool $R_1$
$R_1 \leftarrow \emptyset$ ;                                        ▷ Re-Ranking results
$P \leftarrow R_0$ ;                                                ▷ Re-ranking pool
$F \leftarrow \emptyset$ ;                                          ▷ Graph frontier
while $|R_1| < c$ do
    $B \leftarrow Score($top $b$ from $P$, subject to $c)$ ;        ▷ e.g. monoT5
    $R_1 \leftarrow R_1 \cup B$ ;                                   ▷ Add batch to results
    $R_0 \leftarrow R_0 \setminus B$ ;                              ▷ Discard batch from initial ranking
    $F \leftarrow F \setminus B$ ;                                  ▷ Discard batch from frontier
    $F \leftarrow F \cup (Neighbours(B,G) \setminus R_1)$ ;         ▷ Update frontier
    $P \leftarrow \begin{cases} R_0 & \text{if } P = F \\ F & \text{if } P = R_0 \end{cases}$ ;   ▷ Alternate initial ranking and frontier
end
$R_1 \leftarrow R_1 \cup Backfill(R_0, R_1)$ ;                      ▷ Backfill remaining items
```

Lexically-Accelerated Dense Retrieval (LADR) [17] introduces the concept of adaptive retrieval in an efficient re-ranking setting. LADR uses a lexical initial retriever to select an initial pool of seed documents. From this pool, the method explores neighbouring documents by using a corpus graph based on dense vector similarity, allowing for the retrieval of relevant documents that do not have lexical overlap with the query. They investigate two exploration strategies: Proactive LADR and Adaptive LADR. In Proactive LADR, all neighbouring documents are added to the pool simultaneously, and the entire pool is re-ranked in a single step. Adaptive LADR first scores the initial pool of documents, then retrieves the neighbours of the top-ranked results, scores these new documents, and adds them to the ranking iteratively. Both `LADR` strategies use dense vector similarity to score documents, making it a highly efficient solution.

Query Affinity Modelling (`Quam`) [34] expands on `GAR` by introducing a learned document affinity model. A model that predicts the likelihood that pairs of documents are co-relevant to similar queries, based on training data of query-document pairs with known relevance. This model is used for improved graph construction and a more effective selection of candidate documents. Together, these contributions make `Quam` a highly effective solution.

Recent work demonstrated that Large Language Models (LLMs) are highly effective for document re-ranking in list-wise settings, where multiple documents are ranked together based on their relative relevance [41, 32, 31]. Nonetheless, these approaches are still limited because of the bounded recall problem. Traditional adaptive retrieval methods typically rely on explicit relevance scores to function. `SlideGar` [35] proposes a solution to this limitation. It introduces a sliding window mechanism that iteratively processes a mix of top-ranked documents and new candidates, selected by alternating between documents from the initial ranking and neighbours from the current top-ranked documents. This design enabled adaptive retrieval without relying on explicit relevance scores.

Adaptive retrieval using corpus graphs is quite efficient during query processing time. However, constructing the corpus graph can be computationally expensive, as it requires calculating the similarity between every pair of documents in the corpus. Research [23] demonstrates that approximate methods for building the corpus graph can significantly reduce the construction cost while still achieving good results. Although offline operations may seem less critical to optimise, they can still be valuable in scenarios where these operations must be performed regularly, such as with document corpora that are frequently updated.

In addition to improving overall effectiveness, adaptive retrieval techniques can be leveraged for different tasks. For example, Jaenich et al. [14] demonstrate that adaptive retrieval can improve fair exposure in ranking systems. Their work shows how a corpus graph can be constructed to promote fairness by controlling the diversity of neighbouring documents. Moreover, they propose strategies for adaptively selecting neighbours from pre-existing corpus graphs to encourage diversity. This work highlights the broader potential of adaptive retrieval across various applications.

2.3. Efficient Retrieval

While effectiveness has traditionally been the primary focus in ad-hoc retrieval, the growing scale and complexity of retrieval systems have made efficiency an increasingly important consideration. Efficiency in this context refers to the computational demands of the retrieval process, focusing on minimising query latency and optimising the use of system resources such as processing units and memory usage during search. Latency is significant in real-world applications like web search, where users expect instant responses and systems must process high query volumes over massive document collections. Reducing resource usage reduces operational costs and energy consumption, mitigating the environmental impact of large-scale systems.

In Section 2.1.2, we introduce dense retrieval, which provides significant efficiency gains over cross-encoder-based re-rankers, especially regarding latency. However, many dense retrieval approaches continue to depend on GPUs for query encoding and nearest neighbours search [13, 44, 19]. Nevertheless, dense vector representations provide the foundation for more efficient solutions that reduce resource usage by enabling systems to run on CPUs.

Fast Forward Indexes (`FF`) [18] explore several methods to improve the efficiency of dense re-ranking. Fast Forward uses an index of pre-computed dense vector representations, enabling efficient lookups. Additionally, Fast Forward uses score interpolation, a relatively efficient method of improving effectiveness, combining the advantages of sparse and dense retrieval scores. Furthermore, they explore several techniques to improve the efficiency of the re-ranking pipeline. First, they explore early stopping conditions to avoid unnecessary computations, stopping the process when further lookups are unlikely to affect the top-ranking results. Secondly, they test the use of lightweight query encoders to reduce the computational cost during query processing time by eliminating unnecessary layers and complexity in the query encoder. Finally, Fast Forward explores several methods to reduce the index size, such as sequential coalescing, dimensionality reduction of vector representations, and dynamically removing irrelevant tokens before indexing. Each of these methods reduces either latency or index size, some at the cost of minor decreases in effectiveness.

Cluster-based Partial Dense Retrieval Fused with Sparse retrieval (CDFS) [45] addresses efficient retrieval through a cluster-based approach. CDFS uses a standard clustering algorithm to partition documents into several clusters. It then selects clusters containing the top sparse results from an initial retriever. Additionally, CDFS assigns weights to clusters based on the presence of sparse results, prioritising clusters with a higher concentration of top sparse results for evaluation. CDFS also incorporates score interpolation, combining different types of relevancy signals from both sparse and dense retrievers.

Other research [46] proposes an efficient retrieval system using dense-sparse hybrid vectors within an approximate nearest neighbour (ANN) framework. Unlike typical hybrid approaches that merge sparse and dense results post-retrieval, this method performs unified ANN search directly on hybrid vectors using the HNSW algorithm. However, applying hybrid vectors directly in HNSW presents challenges due to mismatched distance distributions and high computational overhead from sparse components. To address this, the authors introduce a distribution alignment technique to adjust for the imbalance between sparse and dense distance metrics, along with a two-stage indexing strategy that begins with dense vectors and later incorporates sparse components to enhance efficiency while maintaining effectiveness.

3 Methodology

In this section, we begin by elaborating on the main problem our work addresses. We then formulate our research questions in detail, highlighting their relevance and the specific challenges they aim to address. We then present the methodology for answering each question.

3.1. Problem Statement

Balancing effectiveness and efficiency remains a significant challenge in modern information retrieval. This work studies the trade-off between these two aspects in the context of adaptive retrieval. Adaptive retrieval was initially designed to improve the effectiveness of retrieval systems. Nonetheless, its reliance on efficient look-ups in a graph rather than heavy computations or large language models makes it a promising technique for improving effectiveness within efficient pipelines. `LADR` [17] already demonstrates some of the potential of adaptive retrieval in an efficient retrieval context. Our work seeks to build on this research by exploring the efficiency-effectiveness trade-offs in greater depth.

RQ1: How can a corpus graph be used most effectively to overcome the limitations of first-stage lexical retrievers?

The corpus graph offers a promising approach to addressing the limitations of first-stage lexical retrievers, but research on its most effective use remains limited. We seek to determine how relevant neighbours are connected within the corpus graph's structure. We aim to examine the extent to which the clustering hypothesis applies in our settings, specifically whether relevant documents tend to be connected to other relevant ones. Additionally, we aim to improve recall by exploring the most effective strategies for selecting neighbours within a given retrieval budget. Our final objective is to investigate whether adaptive retrieval introduces duplicate documents into the candidate set and explore the most efficient strategy for handling this potential issue.

RQ2: How do scoring mechanisms influence the effectiveness of adaptive retrieval?

Adaptive retrieval aims to overcome recall limitations in retrieval systems but does not directly affect document scoring. We strive to investigate the relationship between adaptive retrieval and scoring mechanisms that produce final rankings. First, we investigate the effect of interpolation-based re-ranking in adaptive retrieval systems. Interpolation-based re-ranking combines scores from both lexical and semantic retrieval systems. Semantically related neighbours to relevant documents may not always share lexical similarities with the queries. As a result, these neighbouring documents might be negatively affected by interpolation-based ranking methods. However, another possibility is that documents sharing semantic and lexical overlap with the query are more likely to be relevant, yielding similar results as interpolation-based retrieval systems without adaptive retrieval. Additionally, we aim to experiment using cross-encoders as an additional phase in our efficient adaptive retrieval setting. We strive to determine whether dense re-ranking limitations restrict adaptive retrieval's potential.

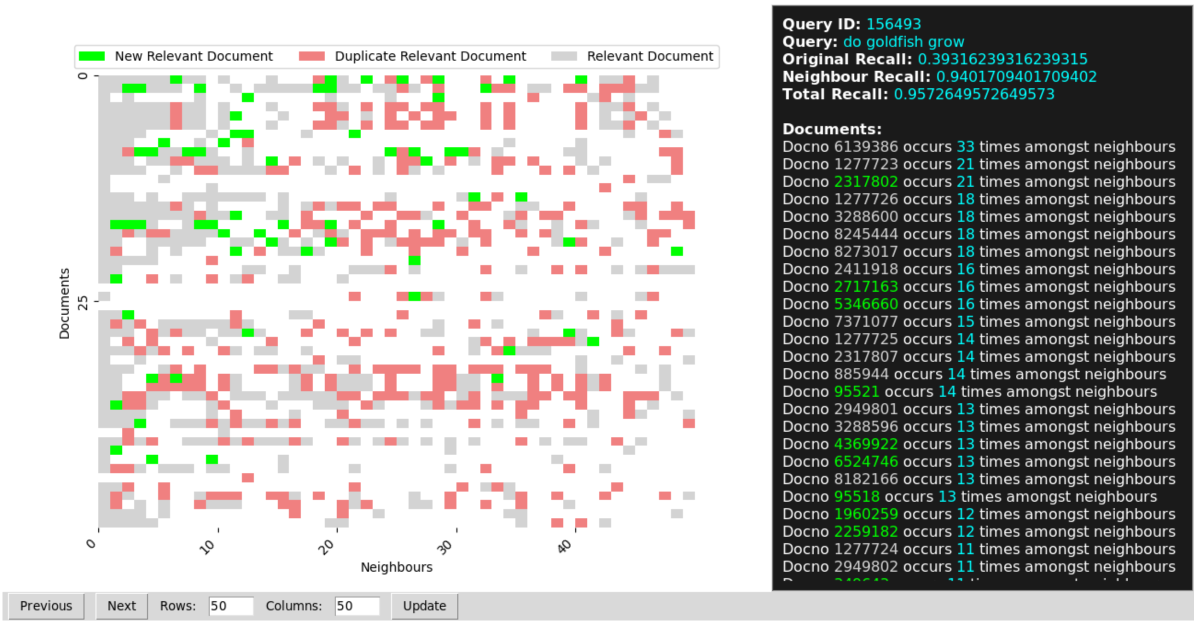
RQ3: How does adaptive retrieval affect efficient re-ranking in effectiveness and efficiency?

Adaptive retrieval seeks to improve the effectiveness of two-stage retrieval systems by increasing the recall of the candidate set. Although adaptive retrieval may introduce additional overhead when generating a candidate set of documents, we aim to experiment with a strategy for handling duplicate documents, which could potentially reduce the number of documents that need to be scored. Considering these different factors, we aim to find the net effect on efficiency and the effectiveness of adaptive retrieval in an efficient re-ranking setting. Additionally, because our cross-encoder-based re-rankers still depend on GPUs and are more challenging to compare with our dense re-ranking systems, we intend to conduct further experiments with these models to enable more accurate comparisons.

3.2. Corpus Graph Analysis

The corpus graph serves as the foundation of current adaptive retrieval methods. This section outlines the process for analysing the most effective use of the corpus graph for efficient adaptive retrieval. The corpus graph contains all documents within a corpus, with edges linking to closely related documents. The clustering hypothesis suggests that relevant documents are more likely to be closely connected to other relevant documents. However, it does not ensure that this is always the case, nor does it guarantee that the initial document is relevant. Given a specific budget, we aim to determine where relevant neighbours are in a corpus graph and the optimal number of neighbours to use.

For our preliminary examination of the corpus graph, we develop a visualisation tool for neighbour analysis named `GarVis`[1] (see Figure 3.1). We use an initial retriever to gather an initial ranking for various queries, then visualise the corresponding neighbours and display several statistics relevant to the graph's structure. All statistics are computed solely over the selected grid.



```
Query ID: 156493
Query: do goldfish grow
Original Recall: 0.39316239316239315
Neighbour Recall: 0.9401709401709402
Total Recall: 0.9572649572649573

Documents:
Docno 6139386 occurs 33 times amongst neighbours
Docno 1277723 occurs 21 times amongst neighbours
Docno 2317802 occurs 21 times amongst neighbours
Docno 1277726 occurs 18 times amongst neighbours
Docno 3288600 occurs 18 times amongst neighbours
Docno 8245444 occurs 18 times amongst neighbours
Docno 8273017 occurs 18 times amongst neighbours
Docno 2411918 occurs 16 times amongst neighbours
Docno 2717163 occurs 16 times amongst neighbours
Docno 5346660 occurs 16 times amongst neighbours
Docno 7371077 occurs 15 times amongst neighbours
Docno 1277725 occurs 14 times amongst neighbours
Docno 2317807 occurs 14 times amongst neighbours
Docno 885944 occurs 14 times amongst neighbours
Docno 95521 occurs 14 times amongst neighbours
Docno 2949801 occurs 13 times amongst neighbours
Docno 3288596 occurs 13 times amongst neighbours
Docno 4369922 occurs 13 times amongst neighbours
Docno 6524746 occurs 13 times amongst neighbours
Docno 8182166 occurs 13 times amongst neighbours
Docno 95518 occurs 13 times amongst neighbours
Docno 1960259 occurs 12 times amongst neighbours
Docno 2259182 occurs 12 times amongst neighbours
Docno 1277724 occurs 11 times amongst neighbours
Docno 2949802 occurs 11 times amongst neighbours
```

Figure 3.1: Example of visualisation created using `GarVis`.

The visualisation is presented as a grid where the first column shows the ranking produced by the initial retriever, with the subsequent columns displaying the neighbours from the corpus graph, ordered by similarity (the most similar documents appearing closest to the left). Each relevant document is labelled by a colour, where green and red documents mean the document is found by adaptive retrieval, and grey means that the document was already present in the initial ranking. Green documents indicate

[1] `https://github.com/martijnsmits/gar_vis`

that it is the first time we encounter the document, determined by the Manhattan distance,

$$d(a,b) = \sum_{i=1}^{n} |a_i - b_i|,$$

from the first document in the initial ranking. Red documents mean we have already encountered the document. Moreover, we show the recall in the original ranking, the recall amongst the neighbours and the total recall that can be achieved using adaptive retrieval. Furthermore, we show how often relevant documents occur amongst the neighbours.

This visualisation allows for exploring corpus graphs through an intuitive, anecdotal method. It provides a visual distribution of neighbouring documents. The key insight we aimed to gain through this visualisation is the relationship between the initially retrieved documents and those found through adaptive retrieval. One of the relationships we explore is the clustering hypothesis. Our goal was to determine if any evidence supports this hypothesis, which would be indicated by rows containing many relevant documents. Another relationship to explore is the location of relevant documents. Specifically, we want to see whether relevant documents are primarily clustered among closely related neighbours or can still be found amongst further neighbours. Finally, relevant documents among neighbours do not necessarily indicate that adaptive retrieval is effective. Therefore, we examine the documents missed by the initial retriever, as only these documents ultimately matter in adaptive retrieval.

While the visualisation provided insight into the relationships between neighbouring documents, its anecdotal nature does not allow us to draw definitive conclusions. Therefore, we conduct a more thorough analysis to generalise these findings beyond exploring individual queries. For a more in-depth graph analysis, we experiment with different configurations of budgets (c), number of neighbours (k) and number of initial documents (s); the relation between these variables is defined as $c = s \cdot (k+1)$, where the term $k+1$ accounts for the initial documents and its k neighbours (see Figure 3.2). The main objective of this analysis is to determine the most effective neighbour depth for efficient adaptive retrieval. We conduct experiments across different budget constraints, experimenting with several different neighbour depths per budget constraint. These experiments aim to discover the optimal configurations for each budget. It should be noted that these configurations are not generalised and may differ for different document corpora.

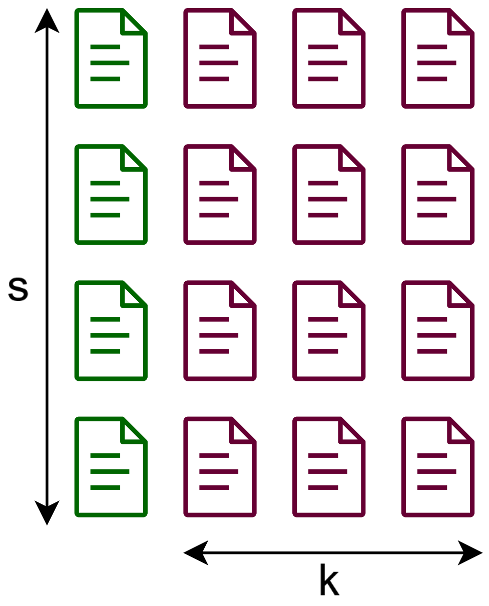


Figure 3.2: Illustration of configurations. Starting with s initially retrieved documents, each contributes its k nearest neighbours, resulting in a candidate set of $c = s \cdot (k+1)$ documents.

In addition, we analyse the effective number of unique documents in our candidate set. Our budget does not always reflect this number due to three key factors. First, many documents share neighbours, leading to significant duplicate documents in the candidate set. Second, the initial retriever may fail to retrieve sufficient documents for specific queries. Lastly, the corpus graph may contain fewer neighbours than the predefined number of neighbours, further limiting the size of our candidate set. As a result, the actual pool of unique documents considered during retrieval can be considerably smaller than the intended budget. Although the mismatch between the budget and the number of unique documents is not inherently problematic, ignoring it can lead to suboptimal retrieval strategies. Re-ranking the same documents multiple times is inherently inefficient. Therefore, it is important to design a strategy that accounts for the potential discrepancy between the effective size and the budget.

3.3. Scoring Mechanisms in Adaptive Retrieval

Fast Forward Adaptive Retrieval (`FFAR`) is a simple adaptive retrieval method built upon the Fast Forward framework [18], though it can be applied to other efficient re-ranking systems. This method is a 1-hop variant of the proactive `LADR` solution [17], enhanced with score interpolation. The idea of `FFAR` is illustrated in Algorithm 2 and Figure 3.3. A sparse retriever retrieves an initial set of candidate documents, which is expanded by adding the top k documents from the corpus graph. Duplicates are removed to avoid scoring the same document multiple times. Fast Forward score interpolation poses an extra challenge, since neighbouring documents do not have sparse scores available. To address this, `FFAR` applies the initial retrieval an additional time to assign sparse scores to the neighbouring documents. The sparse scores are combined with the dense scores from Fast Forward to produce the final ranking.

Algorithm 2: Adaptive Retrieval in an Efficient Re-Ranking System with score interpolation.

Input: Query q, budget c, number of initial results s, k-neighbour corpus graph G_k
Output: Final Ranking R

```
    $R_0 \leftarrow SparseRetrieve(q, s)$ ;                  ▷ e.g. BM25
    $R_1 \leftarrow R_0 \cup Neighbours(R_0, G_k)$ ;        ▷ Add the top $k$ neighbours to the candidate set
    $R_2 \leftarrow RemoveDuplicates(R_1)$ ;                 ▷ Avoid scoring the same document multiple times
    $R_S \leftarrow SparseScore(q, R_2)$ ;                   ▷ e.g. BM25
    $R_D \leftarrow DenseScore(q, R_3)$ ;                    ▷ e.g. Fast Forward indexes with TasB
    $R \leftarrow Interpolate(R_S, R_D)$ ;                   ▷ Interpolate the lexical and dense scores
```

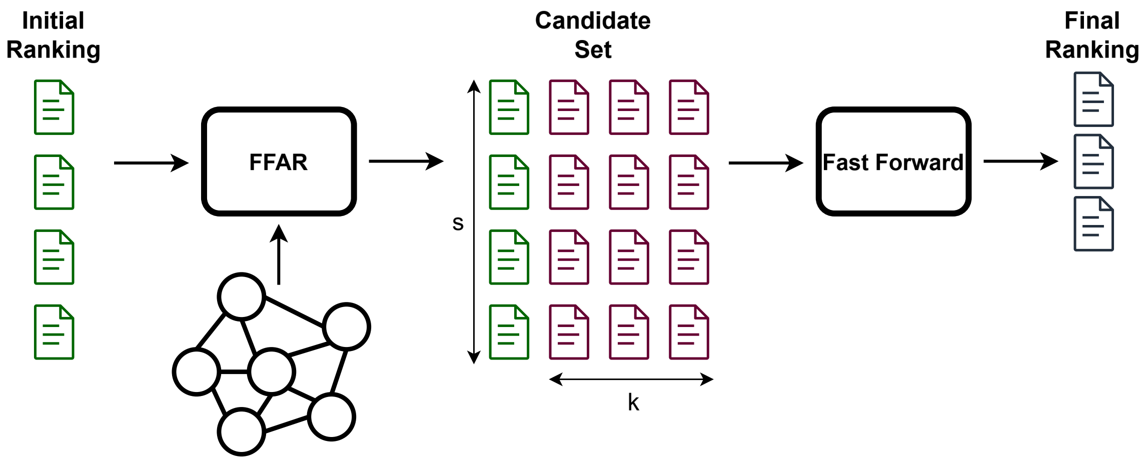


Figure 3.3: Adaptive Retrieval in an Efficient Re-Ranking System with score interpolation.

The first thing we investigate is the impact of score interpolation within an adaptive retrieval setting. To evaluate its impact, we measure the absolute difference in effectiveness between score interpolation and no score interpolation. In the interpolation setting, dense similarity scores are combined with lexical scores, whereas in the non-interpolation setting, the score is composed solely of the dense similarity score. We assess its impact by performing this experiment in our adaptive retrieval setting and our non-adaptive retrieval baseline, and comparing the differences. We calculate the difference by subtracting the effectiveness score of the non-interpolation version from that of the interpolation-based version.

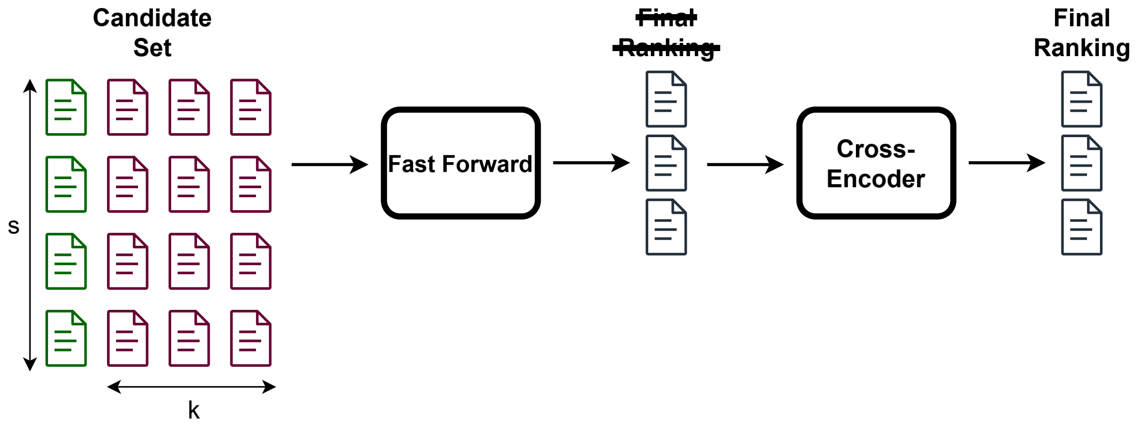


Figure 3.4: Efficient Adaptive Retrieval extended with a Cross-Encoder stage.

We extend our efficient adaptive retrieval setting for our next experiment with an additional cross-encoder stage (see Figure 3.4). In this paper, we explore the trade-off between efficiency and effectiveness. Dual-encoders are generally considered less effective than re-ranking with cross-encoders [38]. We add a cross-encoder stage to our pipeline to investigate whether the improved recall from adaptive retrieval can benefit from better relevance scores. Given the computational cost of cross-encoders, we limit our experiments to smaller cross-encoder budgets. Starting with a large budget would significantly decrease efficiency, as the additional computational cost would outweigh any improvements in effectiveness. Nonetheless, this approach still inherently decreases overall efficiency, as the cross-encoder phase introduces extra computations and requires GPU resources, increasing latency and resource usage.

We refer to this extended version of FFAR with the cross-encoder stage as $\texttt{FFAR}_e$, where e denotes the budget allocated to the cross-encoder stage. As one of our baselines, we also extend LADR to include a cross-encoder stage, which we denote as $\texttt{LADR}_e$. We investigate the impact of the cross-encoder stage across various cross-encoder budgets for different configurations.

3.4. Efficiency in Adaptive Retrieval

So far, our experiments primarily focus on evaluating the effectiveness of retrieval techniques. However, we have not yet conducted any experiments specifically targeting efficiency. We implement the FFAR component of our efficient retrieval pipeline, focusing on minimising unnecessary overhead. We incorporate this component as an intermediary step in the efficient Fast Forward pipeline. We use this efficient pipeline to compare our solution against several different baselines to assess the trade-off between efficiency and effectiveness of each retrieval system.

In the second part of this research question, we compare Quam with $\texttt{FFAR}_e$ and $\texttt{LADR}_e$ using the same cross-encoder budget. Comparing Quam directly with baselines that use dense re-ranking is challenging because these retrieval systems have different hardware requirements and permit much larger re-ranking budgets, which are not feasible for Quam. As a result, we previously used a different configuration for Quam compared to the other baselines, which made the comparison inaccurate. We assess our cross-encoder-based solutions against Quam for a more accurate comparison. We evaluate the effectiveness and efficiency of these retrieval systems across different cross-encoder budgets. Since each retrieval system is allocated the same number of cross-encoder calls, this constant allows us to focus on evaluating the effectiveness and efficiency of the other components of the solutions.

4 Experimental Setup

4.1. Datasets

The **Microsoft MAchine Reading COmprehension (MS MARCO) Passage Re-ranking** document corpus [1] is a large-scale dataset designed to support information retrieval and ranking systems research. It consists of 8.8 million real-world web passages extracted from 3.5 million web documents retrieved by Bing. In addition to the document corpus, MS MARCO Passage offers multiple partitions of queries, each accompanied by relevance labels for associated documents, which are used for training, validation, and evaluation purposes.

We use the **TREC Deep Learning Track** datasets (`TREC DL`) [6, 5, 24] for evaluation. Although MS MARCO Passage includes many queries to support development, a key limitation is that each query is often paired with only one relevant document. This sparsity in relevance labelling can hinder the accurate evaluation of retrieval models. The TREC Deep Learning Track datasets address this limitation by offering an evaluation set for MS MARCO Passage with significantly more relevant documents per query. These relevant documents are human-labelled on the following scale: irrelevant (0), related (1), highly relevant (2) and perfectly relevant (3). The richer annotation helps with the challenges posed by the sparsity of relevant labels in the original MS MARCO Passage dataset. We use the `TREC DL 2019`, `2020` and `Hard` evaluation sets.

In addition to evaluation, we use the TREC Deep Learning `2019` dataset for validation purposes, due to the same concern regarding sparse relevance labels. Specifically, we use it in our corpus graph analysis to determine the optimal configurations for budget and number of neighbours. Although this dataset is mainly used for validation, we still include the `2019` dataset across our other experiments for completeness. Moreover, it should be noted that the `TREC DL Hard` dataset has some overlap in queries with the `TREC DL 2019` dataset.

4.2. Corpus Graph Construction

In our experiments, we construct a corpus graph based on dense similarity scores between documents. Document embeddings are generated using the `TasB` model [13]. Due to the large scale of the corpus, exhaustive similarity computation was not feasible. Instead, we employ an approximate nearest neighbour (ANN) search using the Faiss implementation of the HNSW algorithm [10], which enables efficient similarity search by significantly reducing computation time. Each document is connected to 1024 nearest neighbours based on the resulting similarity search, with graph construction facilitated by the `GAR` library [22]. The general idea of graph construction is demonstrated in Algorithm 3.

Algorithm 3: General idea of Graph Construction

```
Input: Document Corpus $D$, number of neighbours $k$
Output: Corpus Graph $G$
$E \leftarrow \emptyset$ ;                                   ▷ Initialise a set for edges
forall $d_1 \in D$ do
    $R \leftarrow Retrieve(d_1, D) \setminus d_1$ ;          ▷ Retriever/pipeline that produces a score
    $E \leftarrow E \cup \{(d_1, d_2) \mid d_2 \in Top_k(R)\}$ ;   ▷ Add top $k$ edges to E
end
$G \leftarrow Graph(D, E)$
```

4.3. Evaluation Metrics

Metrics are essential for evaluating the effectiveness of an information retrieval system. This research focuses on two different aspects of performance: effectiveness and efficiency. Each requires specific metrics to assess its performance accurately.

Normalised Discounted Cumulative Gain at k (nDCG@k) is our primary evaluation metric for effectiveness (see Equation (4.1)). nDCG is a version of DCG (see Equation (4.2)) where the score is normalised by the IDCG, which represents the DCG of an ideal ranking. nDCG is particularly useful in settings where the top results matter most, controlled by the term $log_2(i+1)$, placing more emphasis on highly ranked documents. Moreover, it also accounts for graded relevance labels, rather than just binary relevance (relevant or not). The parameter k represents the cutoff rank, indicating how many top-ranked results are considered in the evaluation. The TREC DL dataset provides nDCG@10 as one of the official evaluation measures. However, in some experiments, we evaluate with higher values of k to assess performance beyond the top 10 documents.

$$nDCG@k = \frac{DCG@k}{IDCG@k} \tag{4.1}$$

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{log_2(i+1)} \tag{4.2}$$

The (Mean) Reciprocal Rank (RR) is another metric for evaluating effectiveness (see Equation (4.3)). The metric measures how soon we can expect to find a relevant document. This metric is useful for quickly determining the quality of a ranking but lacks some of the nuance provided by the nDCG@k metric, such as graded relevance. The TREC DL also uses this metric as one of the official evaluation measures. Only documents with a relevance label of 2 or higher are considered relevant.

$$RR = \frac{1}{\text{rank of the first relevant document}} \tag{4.3}$$

Another effectiveness metric that we use is the (Mean) Average Precision (AP) (see Equation (4.4)). While precision measures the proportion of relevant documents within a set of retrieved documents, average precision also includes the order of documents in its calculation. Here, n denotes the total number of retrieved documents, i denotes the rank of the current document, $P(i)$ is the precision at cut-off i in the list, and rel_i is zero if the document is not relevant and one if it is relevant. The total number of relevant documents then normalises the sum of these precision values, since the number of retrieved documents may exceed the number of relevant documents. AP is the final official metric used in the TREC DL datasets. Once again, only documents with a relevance label of 2 or higher are considered relevant.

$$AP = \frac{\sum_{i=1}^{n} P(i) \cdot rel_i}{\text{total number of relevant documents}} \tag{4.4}$$

In addition to the previously mentioned evaluation metrics, we also use Recall (R) in some of our experiments (see Equation (4.5)). Recall is particularly important in our context, as the primary goal of adaptive retrieval is to increase the number of relevant documents included in the candidate set. To remain consistent with the TREC DL evaluation standards, we treat only documents with a relevance label of 2 or higher as relevant.

$$Recall = \frac{\text{Relevant Retrieved Documents}}{\text{All Relevant Documents}} \tag{4.5}$$

Regarding efficiency, we evaluate our system's performance based on latency and resource usage. Latency is reported in milliseconds (ms). Resource usage is not reported as an evaluation metric; rather, we distinguish between the use of CPU or GPU, where CPU usage is considered more resource-friendly than GPU usage.

4.4. Baselines

Fast Forward (FF) [18] serves as our baseline for efficient retrieval and forms the foundation of our work. This makes it an ideal point of comparison to evaluate the impact of adaptive retrieval, as the only difference between FF and FFAR is the addition of the adaptive component. FF works by taking a ranking from an initial sparse retriever, encoding the query via $\zeta(q)$, and computing dense relevance scores using pre-computed document embeddings $\eta^{FF}(d)$, resulting in dense score $\phi_D^{FF}(q,d)$ (see Equation (4.6)). These dense scores are then interpolated with sparse scores $\phi_S(q,d)$ from the initial retriever, with the interpolation controlled by a hyperparameter α that is tuned to the dataset and retrieval setup (see Equation (4.7)).

$$\phi_D^{FF}(q,d) = \zeta(q) \cdot \eta^{FF}(d) \tag{4.6}$$

$$\phi(q,d) = \alpha \cdot \phi_S(q,d) + (1-\alpha) \cdot \phi_D^{FF}(q,d) \tag{4.7}$$

Quam [34] serves as our baseline for effectiveness. It builds on GAR [22] by introducing a Learnt Affinity Model designed to capture co-relevance between documents. The model is trained using supervised learning, where it is presented with pairs of documents associated with queries. Positive pairs consist of documents relevant to the query and complementary to each other, while negative pairs include irrelevant documents. By distinguishing between positive and negative examples, the model learns to predict the degree of affinity between document pairs in the context of a given query.

Quam leverages the Learnt Affinity Model for graph construction and set affinity computation. An existing corpus graph provides a basis for graph construction, while edge weights are updated using the affinity model to reflect co-relevance better. Set affinity (see Equation (4.8)) improves candidate selection during re-ranking. Each candidate document is assigned a $SetAff$ score, quantifying its affinity to the top-ranked documents in the re-ranked pool. In this equation, $P(Rel(d'))$ denotes the estimated relevance distribution derived from scoring model ϕ, and $f(d,d')$ measures the affinity between a candidate document d and a top-ranked document d'. Since Quam works differently from efficient retrieval systems, our usual budget settings are not feasible. Therefore, we run several experiments with a configuration of $c = 100$, $s = 30$, and $k = 16$. We also explore additional Quam configurations in a separate experiment to provide a more meaningful comparison.

$$SetAff(d,S) = \sum_{d' \in S} P(Rel(d')) \cdot f(d,d') \tag{4.8}$$

We use LADR [17] as a baseline due to its close similarity to our approach. While LADR offers multiple strategies, we focus on a single variant: proactive LADR (see Figure 4.1) with single-hop neighbours, as it most closely aligns with our approach. This variant works similarly to our solution without score interpolation. It expands the candidate set by selecting the top k documents from the corpus graph and scoring them using dense re-ranking. Additionally, we use LADR with an additional cross-encoder stage for our cross-encoder experiments.

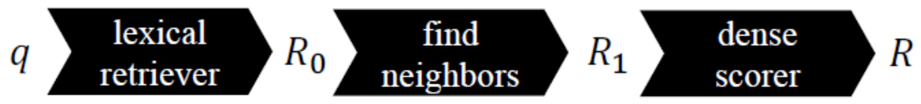


Figure 4.1: Overview of Proactive LADR [17]

Finally, we use Dense Retrieval and Dense Re-ranking as baselines, as these methods use the same embedding models employed in our other solutions, allowing for a consistent comparison across different approaches. We use the dense index, retriever, and scorer from PyTerrier[1]. For Dense Retrieval, we perform an exhaustive search to establish a robust baseline for the effectiveness of the dual-encoder alone. Dense Re-ranking serves as an efficiency baseline.

4.5. Models

We use the BM25 retriever from PyTerrier[2] for our initial retriever. We choose this implementation because it supports re-ranking, which is crucial for score-interpolation in an adaptive retrieval setting. We use the default configuration of the model without modifying any weights.

As our embedding model for dense retrieval and graph construction, we use TasB [13]. TasB uses a dual-teacher approach involving BERT$_{CAT}$ and ColBERT, serving as complementary models for training. BERT$_{CAT}$ uses pairwise teaching for high effectiveness, while ColBERT uses in-batch negative teaching for higher efficiency. This dual-teacher setup allows the model to balance the trade-off between the quality of representations and computational efficiency. For sampling batches, TasB employs two different strategies. The first strategy involves clustering queries by topic using k-means. It samples multiple queries from the same cluster to create a batch, concentrating topic-specific information within a single batch and strengthening the signals for in-batch negative learning. Next, it selects passage pairs to balance the pairwise teacher score margins. While unrelated documents are easy to distinguish, it is much harder to identify non-relevant documents that closely resemble relevant ones. To address this, query and document pairs are grouped by difficulty based on the teacher's scores, ensuring the model is exposed to examples of varying complexity during training. For our experiments, we use a pre-trained model based on a 6-layer DistilBERT encoder with a batch size of 256[3] trained on MSMARCO-Passage.

For our cross-encoder-based experiments, we use MonoT5 [30], a sequence-to-sequence model specifically trained for re-ranking tasks. MonoT5 is built upon the T5 architecture [33], which is widely adopted for various natural language processing tasks due to its flexibility and high performance. MonoT5 takes the query and the document together as input and directly outputs a score indicating how well the document matches the query. For our experiments, we use the base version of MonoT5 with 223 million parameters[4], trained on the MS MARCO dataset.

4.6. Evaluation Setup

Evaluation is performed on the DelftBlue supercomputer [8] offering $17\,000$ CPU cores in over 300 nodes and 80 GPUs over 20 nodes. It provides three types of compute nodes: standard, high memory, and compute nodes with GPUs. We made use of the compute and high-memory nodes equipped with `Intel XEON E5-6248R 24C 3.0GHz` CPUs and GPU nodes with `AMD EPYC 7402 24C 2.80 GHz` CPUs and `NVIDIA Tesla V100S 32GB` GPUs.

For our latency experiments, we ensure all measurements are performed within the same job to minimise the effect of hardware on our experiments. Some retrieval systems require a GPU, while others intentionally run on the CPU to highlight resource efficiency. Therefore, we run all our experiments on a GPU node and explicitly disable the GPU during CPU-based experiments. Each is executed in an isolated Python instance to prevent cross-interference between retrieval systems. For each retrieval

[1]https://pyterrier.readthedocs.io/en/latest/ext/pyterrier-dr/indexing-retrieval.html
[2]https://pyterrier.readthedocs.io/en/latest/terrier-retrieval.html
[3]https://huggingface.co/sebastian-hofstaetter/distilbert-dot-tas_b-b256-msmarco
[4]https://huggingface.co/castorini/monot5-base-msmarco

system, we conduct five runs, each consisting of 10 measurements. We compute each run's average performance, then report the lowest average across five runs. The latency of each retrieval system is measured as the total time taken from the start to the end of the retrieval pipeline.

We tune the α parameter based on Mean Reciprocal Rank (MRR) using a subset of 500 queries from the MSMARCO Passage Small development set. For our retrieval setup consisting of a PyTerrier `BM25` retriever and a `TasB` embedding model, we identify $\alpha = 0.1$ as an effective value for the MSMARCO Passage corpus.

4.7. Configurations

In Section 3.2, we introduce the concept of configurations. Each configuration begins with s initially retrieved documents, where each document contributes its k nearest neighbours to form the candidate set. As a result, the candidate set contains $c = s \cdot (k+1)$ documents. We design our configurations to align with several fixed budgets, enabling direct comparisons between configurations based on budget. This section details the construction process, ensuring that all budgets are comparable and each parameter is a natural number.

Construction

We choose the values for c and k, which determine our value of s together. We construct our configurations in a way such that the following equation holds:

$$s = \frac{c}{k+1}, \quad \text{where} \quad s \in \mathbb{N} \tag{4.9}$$

To ensure this condition is satisfied, we define the parameters k and c as follows:

$$k = 2^i - 1 \quad \Rightarrow \quad k + 1 = 2^i, \quad \text{where} \quad i \in \{0, 1, ..., n\}$$

and

$$c = j \cdot 2^n, \quad \text{where} \quad j \in \{1, 2, ..., m\}$$

Verification

Now we verify that Equation (4.9) holds with these chosen values of k and c. For s to be in $\mathbb{N}$, the following condition must hold:

$$c \equiv 0 \pmod{k+1}$$

First, we express n in terms of i and some natural number x:

$$n = i + x, \quad \text{where} \quad x \in \mathbb{N}$$

We can rewrite our equation for c as follows:

$$\begin{aligned} c &= j \cdot 2^n \\ &= j \cdot 2^{i+x} \\ &= j \cdot 2^i \cdot 2^x \\ &= (j \cdot 2^x) \cdot 2^i \end{aligned}$$

Now we examine the modulo condition:

$$(j \cdot 2^x) \cdot 2^i \equiv 0 \pmod{2^i}$$

This holds because $j \cdot 2^x$ is a natural number, and 2^i divides $(j \cdot 2^x) \cdot 2^i$ by construction. Therefore, our condition holds with these values of k and c $\square$.

4.8. Fast Forward Adaptive Retrieval

We propose Fast Forward Adaptive Retrieval[5] (`FFAR`) as an adaptive retrieval component for efficient retrieval pipelines. `FFAR` functions as a PyTerrier Transformer[6] that appends neighbouring documents to the initial ranking. The PyTerrier Transformer, not to be confused with the Transformer architecture, accepts incoming data, applies a transformation, and passes it on to the next stage. These transformers are often compatible with other transformers to form a re-ranking pipeline. The incoming and outgoing data depend on the pipeline stage, starting from queries and ending with a ranking of documents for each query. `FFAR` takes a set of queries and documents as input and transforms the data by appending the top k nearest neighbours for each document. Score interpolation presents an additional challenge because it requires lexical scores for all documents, including the newly added neighbours, for which lexical scores are not initially computed. To address this, we introduce an extra BM25 scoring step to compute lexical scores for the neighbours. The resulting scored documents are then passed to the next stage, where Fast Forward is used to produce final scores. Our `FFAR` components do not depend on Fast Forward and can be integrated into any scoring pipeline that accepts queries and documents as input.

[5]https://github.com/martijnsmits/ffar

[6]https://pyterrier.readthedocs.io/en/latest/transformer.html

5

Results and Discussion

5.1. Corpus Graph Analysis

RQ1: How can a corpus graph be used most effectively to overcome the limitations of first-stage lexical retrievers?

To answer this question, we first explored neighbourhoods using our visualisation tool: `GarVis`. Next, we analyse different configurations of budgets (c), number of neighbours (k) and number of initial documents (s). We evaluate the different configurations based on recall, eliminating the effect of scoring mechanisms influencing the evaluation of our different configurations. Recall provides a straightforward method to assess how many relevant documents were included in our candidate set, aligning directly with the ultimate goal of adaptive retrieval. We explore up to 1023 neighbours and a budget of up to $16\,384$. Finally, we analyse the presence of duplicate documents in these configurations.

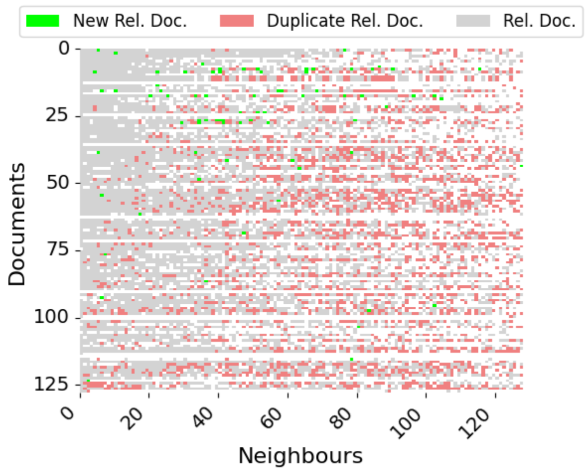
Neighbourhood Visualisation

First, we use `GarVis` to explore the corpus graph and gain an initial understanding of the relations between documents. We use `GarVis` to analyse neighbourhood patterns on a per-query basis. First, we analyse document neighbourhoods to explore potential recall gains through adaptive retrieval. Table 5.1 shows improvements in recall on a per-query basis. This experiment is conducted with a budget of $16\,384$ and 127 neighbours, with recall improvements measured relative to the initial ranking of 128 documents. We observe recall improvements in 37 of the 43 queries using neighbouring documents to expand the candidate set. Among these, 12 achieve perfect recall. For the remaining six queries, we observe no improvement. Five of these already had perfect recall initially, and in the other case, the initial retriever failed to retrieve any relevant documents. Naturally, recall can only improve or remain the same since we include the 128 results from our initial retriever in the computation of our total recall. In this experiment, we do not focus on whether adaptive retrieval can match the initial retriever with similar budgets, but rather on the ability of the corpus graph to provide additional relevant documents. The results show that adaptive retrieval has potential, allowing us to find additional relevant documents.

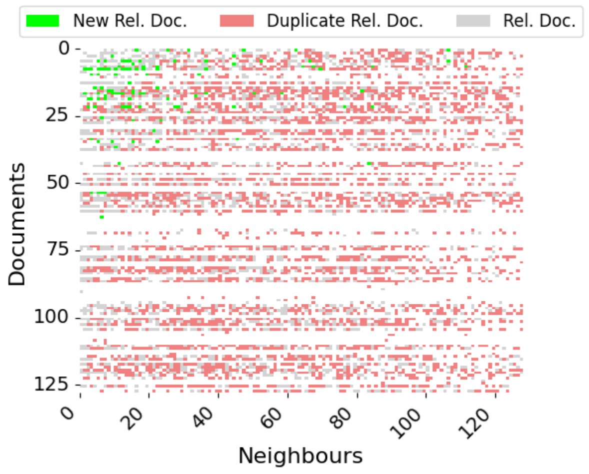
Result Type	Number of Queries
Improvement	**37**
– Reached Perfect Recall	12
– Increased Recall	25
No Improvement	**6**
– Already Perfect Recall	5
– Started from Zero Recall	1

Table 5.1: Recall improvements evaluated on a per-query basis across 43 queries from the `TREC DL 2019` dataset.

Next, we focus on how neighbouring documents relate to one another. We find patterns similar to those shown in Figure 5.1 for several queries. For these queries, our initial retriever finds many relevant documents, along with many relevant neighbour documents. We can see that there are still several new documents to be found in these cases. However, this does not necessarily mean that the initial retriever would not find these documents if we run it at a higher retrieval depth. We use `GarVis` to increase the retrieval depth for queries to $16\,384$, matching the adaptive retrieval budget of the candidate set. For queries `168216` and `264014`, we achieve recalls of 1 and 0.99, respectively, higher than the recall achieved by adaptive retrieval on the same budget. We apply the same approach to other queries with similar neighbourhoods and observe the same effect.



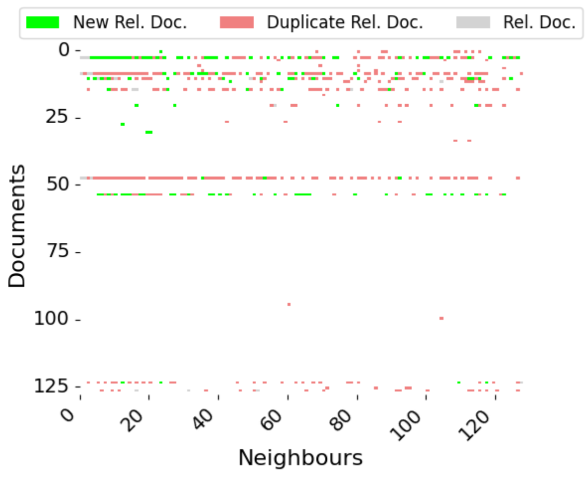
(a) Query `168216`: *does legionella pneumophila cause pneumonia*



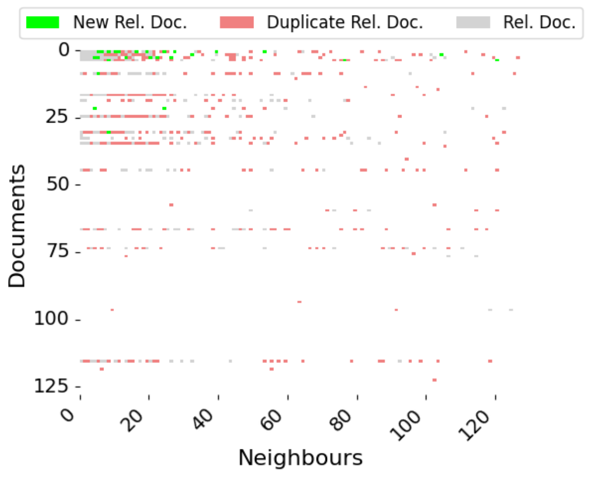
(b) Query `264014`: *how long is life cycle of flea*

Figure 5.1: Neighbourhoods where the initial retriever finds many relevant documents.

Another pattern we find several times is where the initial retriever finds fewer relevant documents (see Figure 5.2). For these queries, we see that despite the few relevant documents from the initial ranking, we quickly find many additional relevant documents through neighbouring documents. We also increase the retrieval depth for these queries to $16\,384$. At this depth, we achieve only a recall of 0.10 for query `1063750`, while adaptive retrieval at the same budget achieves a recall of 0.74. For query `833860`, the initial retriever alone achieves a recall of 0.81, whereas adaptive retrieval reaches a recall of 0.98 at the same budget.



(a) Query `1063750`: *why did the us volunterilay enter ww1*



(b) Query `833860`: *what is the most popular food in switzerland*

Figure 5.2: Neighbourhoods where the initial retriever finds few relevant documents.

Our initial exploration with `GarVis` suggests that the effect of adaptive retrieval varies. While the effectiveness for some queries significantly improves, adaptive retrieval may reduce effectiveness for others. We observe a pattern where adaptive retrieval reduces effectiveness for queries where the initial retriever retrieves many relevant documents, but improves effectiveness for queries where fewer relevant documents are retrieved. However, this exploration of neighbourhoods is anecdotal, and this behaviour may vary across different queries. Furthermore, we observe a pattern where relevant documents are more frequently related to other relevant documents, supporting the clustering hypothesis and suggesting that it holds to a certain extent.

Configuration Analysis

For a more generalised analysis of the corpus graph, we evaluate different configurations, each defined by a combination of budget and number of neighbours. Figure 5.3 presents the results for various smaller budgets. There are no observed improvements in recall at these lower budget levels, with a budget of 8192 being the first to achieve recall comparable to the baseline. What is interesting to see is that we do observe improvements in terms of nDCG@10 for the majority of the configurations, despite the lack of improvements in recall. Using 511 neighbours or more at these budgets leads to a drop in recall and nDCG@10.

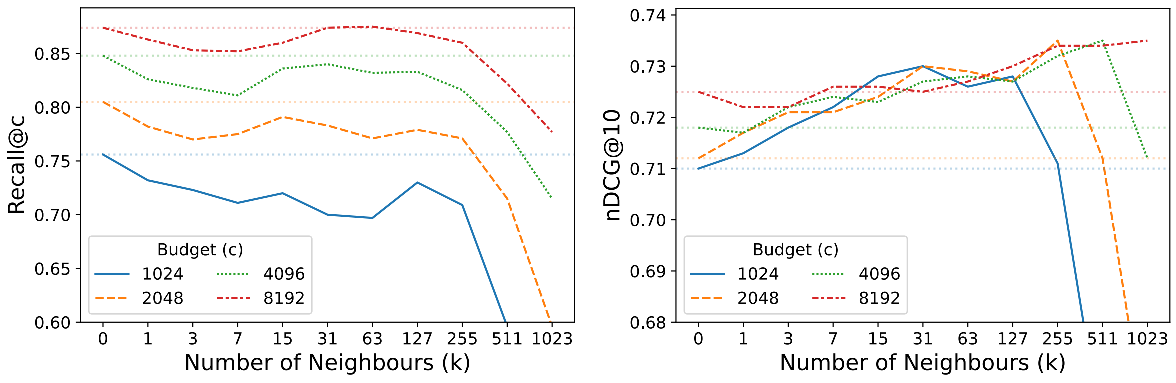


Figure 5.3: Effectiveness across varying configurations for lower budgets on the `TREC DL 2019` dataset.

Figure 5.4 shows the results for larger budgets. Unlike the previous results with smaller budgets, where we observed no recall improvement, we now see an increase in recall with the larger budgets. For the three smallest budgets in this graph, the best recall occurs at 63 neighbours, while for the two largest budgets, the best recall is at 127 neighbours. Interestingly, while recall decreases with more neighbours, the nDCG@10 improves as the number of neighbours increases.

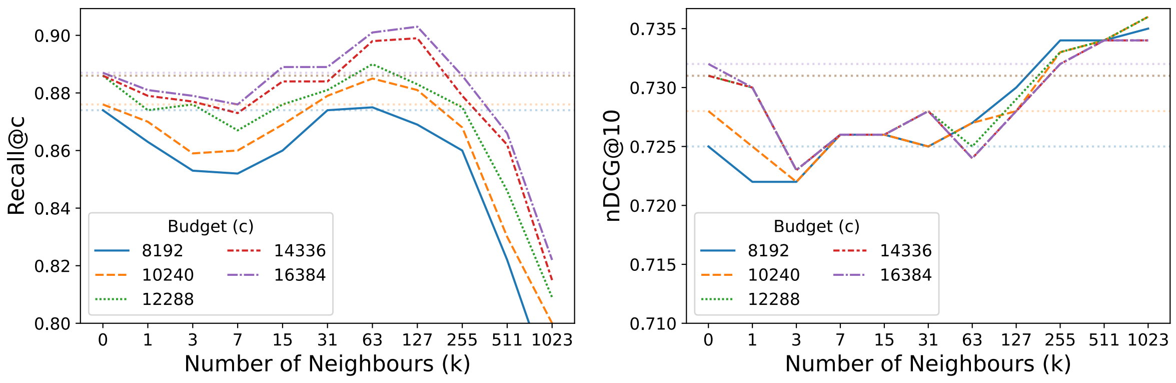


Figure 5.4: Effectiveness across varying configurations for higher budgets on the *TREC DL 2019* dataset.

One observation is the difference in effectiveness between our smaller and larger budgets. Larger budgets have more to gain from adaptive retrieval than smaller ones. In efficient retrieval settings, document scoring is relatively inexpensive, so ranking costs have less impact on the efficiency of the overall pipeline. As a result, the lower effectiveness of dense similarity can be offset by allocating larger re-ranking budgets. As a result, efficient pipelines have less to gain from careful candidate selection, such as the methods proposed by Quam, compared to cross-encoder-based approaches, where document scoring is more expensive. In lower budget settings, it seems more effective to re-rank a greater number of initial documents, since relevant ones are still likely to be included in the initial ranking. With higher budgets, the initial ranking seems to yield diminishing returns regarding relevant documents, meaning we start to gain more by including neighbouring documents in our candidate set.

We find the highest recall for high budgets at 63 or 127 neighbours, depending on the budget. These findings show a difference in the number of neighbours used in various retrieval strategies. While effectiveness-driven methods like Quam and GAR experiment with a smaller number of neighbours (up to 16), the more efficient baseline, LADR, achieves good results with a larger number of neighbours, similar to our findings. This suggests that in efficient pipelines, where re-ranking budgets tend to be much larger, using more neighbours seems more advantageous.

Next, recall and nDCG@10 do not necessarily correlate across different configurations. High recall does not necessarily guarantee that relevant documents are ranked highly. While adaptive retrieval focuses on increasing recall within the candidate set, the scoring mechanisms ultimately determine the final ranking. This highlights how the choice of scorer can significantly impact the overall effectiveness of a system. We conduct additional experiments to explore the effects of scoring mechanisms and present our findings in Section 5.2.

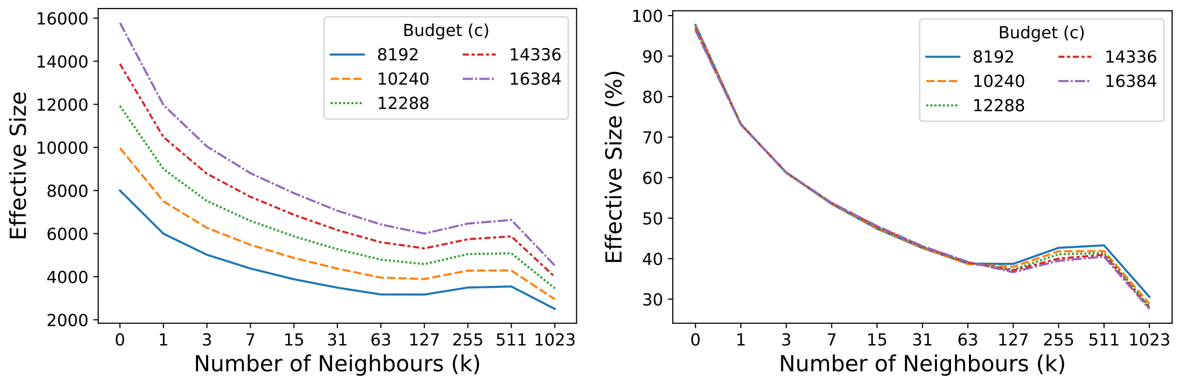
Based on these results, we select a configuration for each larger budget according to the highest recall effectiveness. While nDCG@10 is the primary metric for effectiveness, we deliberately prioritise recall in our decision-making. The first reason for this is that the goal of adaptive retrieval is to improve the recall of the candidate set. The second reason is that choosing configurations based on nDCG@10 could lead to overfitting to our dataset. It is important to note that these configurations may not universally apply to all datasets; they have been specifically tuned for the MSMARCO Passage dataset. The selected configurations are listed in Table 5.2. From now on, any mention of a budget refers to the corresponding configuration of the number of neighbours.

Budget (c)	**Number of neighbours (k)**
8192	63
10240	63
12288	63
14336	127
16384	127

Table 5.2: The selected configurations.

Duplicate Analysis

The last part of this research question focuses on duplicate documents. Figure 5.5 shows the candidate size without duplicates. It shows that for each budget, the behaviour is generally consistent: the effective size of the candidate set drops sharply at first, then the decline becomes less steep as more neighbours are added. There is a small local peak around 511 neighbours, followed by a further decrease to the lowest effective size at 1023. Notably, the lowest effective size is observed for the best configurations, with the effective size at approximately 40% of the total re-ranking budget.



(a) Effective size of the candidate set. **(b)** Effective size relative to the full budget.

Figure 5.5: Effective size of the candidate set after de-duplication per budget on the `TREC DL 2019` dataset.

The presence of duplicate documents is expected, as (1) documents share neighbours, (2) initial retrievers do not always retrieve sufficient documents for specific queries, and (3) the corpus graph does not always contain enough neighbours. While the degree of decrease in the size of the candidate set is not unexpected, it is unusual that the most effective configurations occur at one of the lowest effective sizes. One possible explanation is that these configurations exhibit the most overlap in neighbours, resulting in a candidate set of closely related documents. However, this could also be an anomaly in the data.

The small local peak at 511 neighbours might seem unusual initially, but it is likely explained by the fact that each document exclusively contains unique neighbours. Consequently, large numbers of neighbours can lead to a spike of this nature. However, we do not observe this effect with smaller numbers of neighbours because the number of shared neighbours between documents likely plays a larger role in the effective size of the candidate set. The absence of a similar effect at 1023 neighbours is likely because many documents do not have that many neighbours. As a result, the initial set of retrieved documents is reduced by half, while the gain from additional neighbours does not sufficiently offset this reduction.

The results reveal a significant presence of duplicate documents in the candidate set. The next step is to determine a strategy to address this issue. One potential approach is adding documents until we satisfy the re-ranking budget. However, the challenge with this solution lies in determining when to stop adding documents, as newly added documents may also be duplicates. Any strategy involving the addition of documents introduces additional overhead and is likely inefficient. Moreover, the relevance of new documents likely decreases as they move further from the original relevant seed documents. Instead, we remove duplicate documents from the candidate set, as it introduces relatively low additional overhead. This also offers the added benefit of reducing the candidate set, meaning fewer documents must be re-ranked, resulting in lower latency and resource consumption in the re-ranking step.

5.2. Scoring Mechanisms in Adaptive Retrieval

RQ2: How do scoring mechanisms influence the effectiveness of adaptive retrieval?

Adaptive retrieval is a technique designed to improve the effectiveness of retrieval systems. However, adaptive retrieval does not directly score the documents. Instead, it increases the recall within the candidate set. The underlying idea is that higher recall increases the chances of ranking more relevant documents near the top. We evaluate the effect of score interpolation and cross-encoders on the effectiveness of an efficient adaptive retrieval pipeline.

Score Interpolation

The first scoring mechanism that we evaluate is score interpolation. Score interpolation combines the scores of the initial retriever and the re-ranker into one score (see Equation (4.7)). The results for this experiment can be found in Table 5.3 and Table 5.4. We see that score interpolation positively affects both Fast Forward and FFAR. On the TREC DL 2019 dataset, FFAR generally benefits more from interpolation than Fast Forward. This trend is even more pronounced in the TREC DL 2020 dataset, where FFAR consistently benefits more than Fast Forward. The TREC DL 2019 dataset shows comparable improvements across all retrieval depths for both systems. The TREC DL 2020 dataset shows the greatest improvements at a retrieval depth of 50 documents. Finally, on the 2019 dataset, both retrieval systems achieve comparable improvements in recall at a retrieval depth of 100, with FFAR showing a slight advantage. In contrast, on the 2020 dataset, FFAR continues to improve recall at depth 100, while Fast Forward shows little to no improvement.

Budget (c)	**Retriever**	TREC DL 2019			
		nDCG@10	*nDCG@50*	*nDCG@100*	*R@100*
8192	FF	0.725 (0.010▲)	0.666 (0.009▲)	0.659 (**0.012**▲)	0.618 (**0.010**▲)
	FFAR	0.727 (**0.015**▲)	0.662 (**0.013**▲)	0.654 (0.011▲)	0.625 (0.007▲)
10240	FF	0.728 (0.010▲)	0.668 (0.009▲)	0.661 (0.012▲)	0.618 (0.010▲)
	FFAR	0.727 (**0.015**▲)	0.661 (**0.012**▲)	0.654 (**0.013**▲)	0.625 (**0.011**▲)
12288	FF	0.731 (0.013▲)	0.671 (0.009▲)	0.663 (0.012▲)	0.624 (0.008▲)
	FFAR	0.725 (**0.018**▲)	0.662 (**0.013**▲)	0.655 (**0.015**▲)	0.625 (**0.010**▲)
14336	FF	0.731 (0.014▲)	0.671 (0.010▲)	0.663 (0.012▲)	0.624 (0.008▲)
	FFAR	0.728 (0.014▲)	0.664 (**0.014**▲)	0.656 (**0.016**▲)	0.624 (**0.014**▲)
16384	FF	0.732 (0.013▲)	0.671 (0.011▲)	0.663 (0.013▲)	0.624 (0.009▲)
	FFAR	0.728 (**0.014**▲)	0.663 (**0.014**▲)	0.655 (**0.016**▲)	0.622 (**0.014**▲)

Table 5.3: Effectiveness improvements through interpolation on TREC DL 2019.

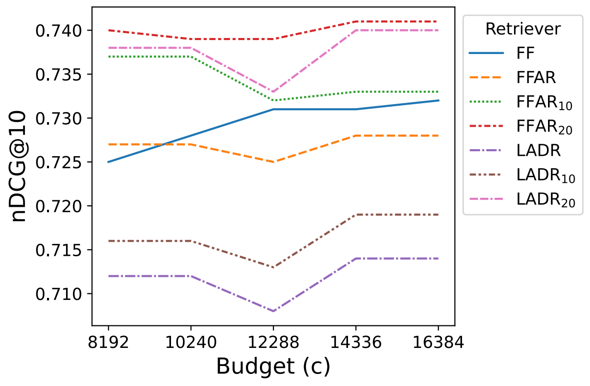
Budget (c)	**Retriever**	TREC DL 2020			
		nDCG@10	*nDCG@50*	*nDCG@100*	*R@100*
8192	FF	0.709 (0.017▲)	0.668 (0.020▲)	0.665 (0.015▲)	0.738 (0.000)
	FFAR	0.704 (**0.021**▲)	0.660 (**0.028**▲)	0.658 (**0.023**▲)	0.724 (**0.009**▲)
10240	FF	0.708 (0.018▲)	0.668 (0.021▲)	0.666 (0.016▲)	0.739 (0.001▲)
	FFAR	0.705 (**0.021**▲)	0.663 (**0.030**▲)	0.661 (**0.024**▲)	0.734 (**0.011**▲)
12288	FF	0.708 (0.017▲)	0.667 (0.021▲)	0.666 (0.017▲)	0.739 (0.001▲)
	FFAR	0.705 (**0.022**▲)	0.662 (**0.029**▲)	0.661 (**0.024**▲)	0.734 (**0.011**▲)
14336	FF	0.708 (0.017▲)	0.666 (0.021▲)	0.665 (0.017▲)	0.739 (0.001▲)
	FFAR	0.704 (**0.022**▲)	0.659 (**0.030**▲)	0.659 (**0.026**▲)	0.734 (**0.017**▲)
16384	FF	0.708 (0.017▲)	0.666 (0.022▲)	0.665 (0.017▲)	0.739 (0.001▲)
	FFAR	0.704 (**0.022**▲)	0.660 (**0.030**▲)	0.660 (**0.026**▲)	0.734 (**0.017**▲)

Table 5.4: Effectiveness improvements through interpolation on TREC DL 2020.

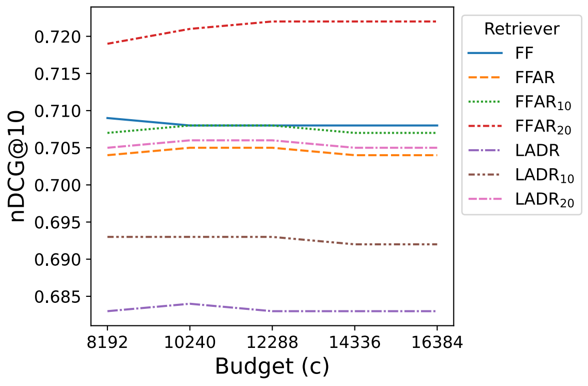
Adaptive retrieval is typically used to identify documents that the initial retriever missed, meaning these documents usually receive lower sparse scores than those retrieved in the first pass. One possibility is that score interpolation negatively affects the scores of these documents. Nonetheless, the results indicate that interpolation yields an even more substantial positive effect when used alongside adaptive retrieval. These results indicate that score interpolation serves its intended purpose by combining the complementary signals from both retrieval approaches. These results suggest that documents sharing substantial lexical and semantic similarity to the document are more likely to be relevant than documents sharing similarity in just one aspect. It is important to note that documents retrieved through adaptive retrieval can still achieve high rankings, as they can be more semantically similar to the query than those retrieved by the initial retriever.

Cross-encoders

Next, we experiment with cross-encoders in the efficient adaptive retrieval pipeline. Figure 5.6 shows the results of these experiments. Both datasets show that $\texttt{FFAR}_{20}$ has the highest overall results in effectiveness. We observe that cross-encoders show higher effectiveness than their base versions, with increased cross-encoder budget leading to increased effectiveness. Moreover, FFAR variants demonstrate higher effectiveness than their LADR counterparts. Fast Forward (FF) performs well, outperforming FFAR, LADR, and $\texttt{LADR}_{10}$ on the 2019 dataset, and showing comparable performance to $\texttt{FFAR}_{10}$ on the 2020 dataset."



(a) TREC DL 2019



(b) texttt TREC DL 2020

Figure 5.6: Effectiveness of cross-encoder calls on FFAR and LADR.

The results show that using an additional cross-encoder stage improves the effectiveness of our efficient adaptive retrieval systems. The selected cross-encoder budgets are relatively small, meaning that with a limited number of cross-encoder calls, we can achieve noticeable improvements in effectiveness. In Section 5.3, we build on these findings by incorporating latency into our analysis, allowing us to evaluate how the cross-encoders impact the efficiency of our retrieval systems.

Moreover, FFAR variants outperform their LADR counterparts, which is consistent with our interpolation-based experiments, as interpolation is the distinguishing factor between FFAR and LADR. However, $\texttt{LADR}_{20}$ performs notably well on the 2019 dataset, indicating that even a relatively small cross-encoder budget can sometimes surpass the benefits of interpolation-based re-ranking.

5.3. Efficiency in Adaptive Retrieval

RQ3: How does adaptive retrieval affect efficient re-ranking in effectiveness and efficiency?

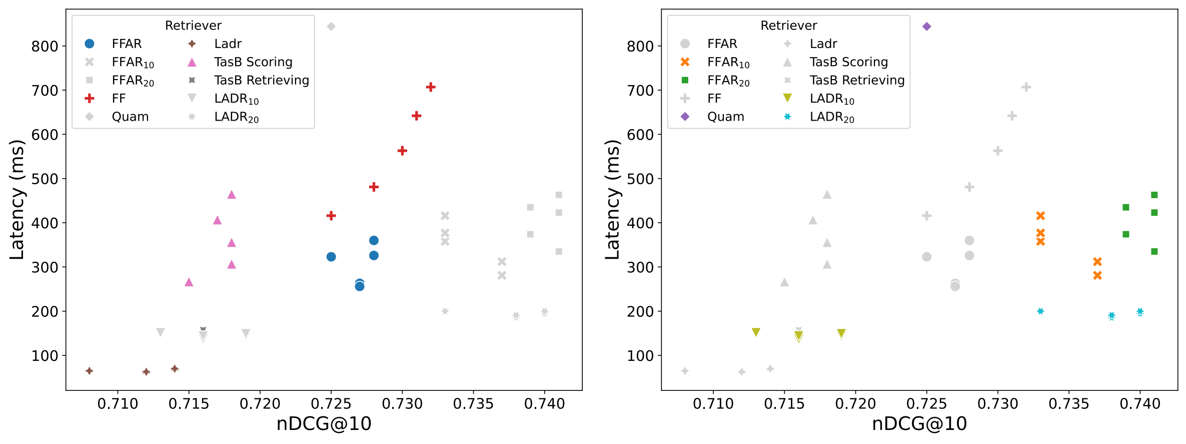
A key focus of this research is the balance between efficiency and effectiveness. In this section, we present our findings on this trade-off. First, we provide a general comparison of our solution against the baselines. Then, in the second part, we take a closer look at the efficiency-effectiveness trade-off, specifically within our cross-encoder-based solutions.

Balancing Efficiency and Effectiveness

The results for this part of the research question can be found in Table 5.5 and Figure 5.7 for the TREC DL 2019 dataset, Table 5.6 and Figure 5.8 for the TREC DL 2020 dataset, and Table 5.7 and Figure 5.9 for the TREC DL Hard dataset. Since we use the TREC DL 2019 as the validation set, which overlaps with TREC DL Hard, we treat TREC DL 2020 as our primary evaluation set and compute statistical significance based on this dataset. The tables contain our metrics for evaluating the retrieval systems, with nDCG@10 as our primary metric for effectiveness and latency as our main metric for efficiency (see Section 4.6 for our approach to measuring latency). We also include RR and AP@c, as they are official metrics for these evaluation datasets. In some cases, we do not report AP@c because specific retrieval systems do not retain the full ranking, and we prefer not to modify these implementations in a way that could negatively impact efficiency. Instead, we include AP@10 as an additional metric. In our figures, we display all the datapoints on the same graph.

Retriever	**Device**	TREC DL 2019				
		nDCG@10	*RR*	*AP@10*	*AP@c*	*Latency (ms)*
BM25	***CPU***	0.480	0.642	0.108	0.289	-
Fast Forward	***CPU***	0.725	0.869	0.221	0.486	416
TasB Retrieval	***CPU***	0.716	**0.886**	0.219	0.462	158
TasB Scoring	***CPU***	0.715	0.864	0.215	0.478	267
Quam	*GPU*	0.725	0.875	0.22	0.421	844
LADR	***CPU***	0.712	0.868	0.216	0.478	**60**
LADR_{10} (MonoT5)	*GPU*	0.716	0.829	0.216	-	137
LADR_{20} (MonoT5)	*GPU*	0.738	0.849	0.228	-	186
FFAR	***CPU***	0.727	0.868	0.224	**0.487**	263
FFAR_{10} (MonoT5)	*GPU*	0.737	0.856	**0.231**	-	281
FFAR_{20} (MonoT5)	*GPU*	**0.741**	0.865	0.224	-	335

Table 5.5: Retrieval performance in effectiveness and efficiency on (TREC DL 2019) for retrieval budget $c = 8192$.



(a) CPU-based systems highlighted.

(b) GPU-based systems highlighted.

Figure 5.7: Comparison of effectiveness against efficiency on (TREC DL 2019). Multiple different entries for retrieval systems are for different budgets.

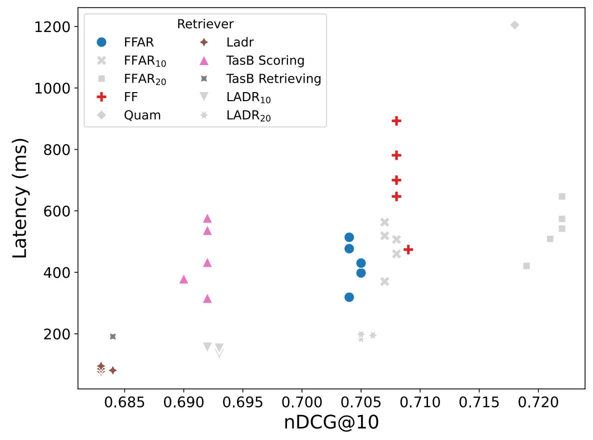
For our CPU-based retrieval systems, the results are fairly consistent across all three datasets. LADR is a highly efficient baseline but lacks effectiveness. On average, TasB retrieval achieves effectiveness similar to LADR, though it is generally slower. Regarding TasB scoring, latency is comparable to FFAR, but its effectiveness is lower than that of FFAR. Overall, while FFAR tends to have lower effectiveness than FF in most cases, it is more efficient across the board. FF typically appears to be the most effective CPU-based retrieval system, but also the least efficient.

We see that FFAR_{20} is the most effective system on the 2019 and 2020 datasets, while Quam shows the highest effectiveness of all systems on the Hard dataset. For our FFAR_e and LADR_e experiments, we see that the versions with 20 cross-encoder calls, in most cases, perform better than the versions with 10 cross-encoder calls, while latency increases. The only exception is the Hard dataset, where both FFAR_e systems have similar effectiveness. While LADR_{10} shows the highest efficiency of all GPU-based systems, it is also the least effective. LADR_{20} generally performs well in effectiveness while remaining amongst the more efficient retrieval systems.

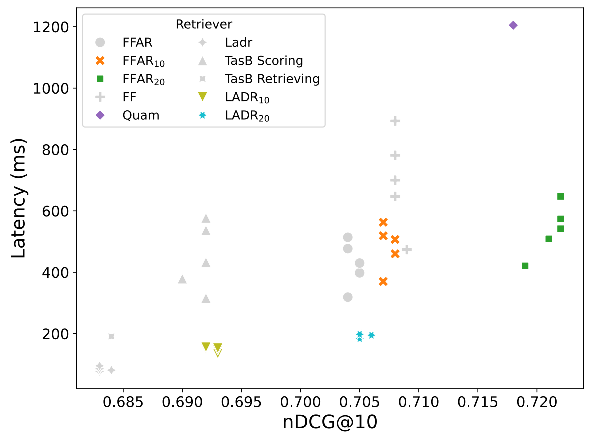
Overall, our FFAR-based systems outperform the LADR-based systems in terms of effectiveness, while LADR leads in efficiency. The improved effectiveness is not unexpected, as our previous results indicate that score interpolation positively impacts system effectiveness. However, seeing how much more efficient the LADR-based systems are is surprising. Although the FFAR systems include an additional lexical scoring phase and score interpolation, these extra operations are unlikely to account for the

Retriever	**Device**	nDCG@10	RR	AP@10	AP@c	Latency (ms)
		TREC DL 2020				
BM25	***CPU***	0.494^{1-3}	0.619	0.183	0.294	-
Fast Forward	***CPU***	0.709	0.857	**0.314**	**0.497**	474
TasB Retrieval	***CPU***	0.684	0.824	0.296	0.468	191
TasB Scoring	***CPU***	0.692	0.825	0.302	0.481	316
Quam	*GPU*	0.718	**0.875**	0.309	0.467	1205
LADR	***CPU***	0.683	0.833	0.298	0.468	**77**
LADR_{10} (MonoT5)	*GPU*	0.693	0.84	0.303	-	135
LADR_{20} (MonoT5)	*GPU*	0.705	0.818	0.307	-	184
^{1}FFAR	***CPU***	0.704	0.857	**0.314**	0.489	319
$^{2}\text{FFAR}_{10}$ (MonoT5)	*GPU*	0.707	0.842	0.311	-	370
$^{3}\text{FFAR}_{20}$ (MonoT5)	*GPU*	**0.719**	0.854	**0.314**	-	421

Table 5.6: Retrieval performance in effectiveness and efficiency on (TREC DL 2020) for retrieval budget $c = 8192$. Superscripts indicate statistical significance using a two-sided paired test with a 95% confidence level, applying the Bonferroni correction.



(a) CPU-based systems highlighted.



(b) GPU-based systems highlighted.

Figure 5.8: Comparison of effectiveness against efficiency on (TREC DL 2020). Multiple different entries for retrieval systems are for different budgets.

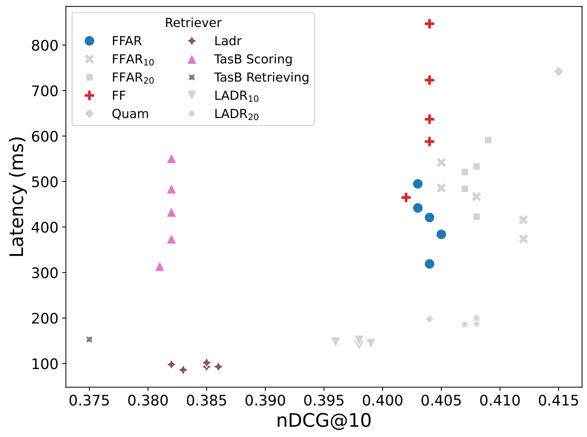
degree of increased latency observed in these systems. Different implementations of the systems likely explain the difference in this latency. Overall, LADR_{20} seems to be the retriever that best balances efficiency and effectiveness. Efficiently integrating score interpolation into the LADR implementation could result in a highly competitive solution for efficient retrieval.

Another interesting observation is that adaptive retrieval-based systems are generally more efficient yet less effective than their re-rankers without adaptive retrieval. Adaptive retrieval improves effectiveness while introducing additional overhead. Therefore, it is strange to see the opposite effect in our results. Our earlier results found that adaptive retrieval introduces many duplicate documents in the candidate set. The strategy to deal with this issue was to remove all duplicate documents from our candidate set. These results demonstrate that while a smaller candidate set leads to lower effectiveness, the overall latency is reduced because significantly fewer documents are scored despite the additional overhead from adaptive retrieval.

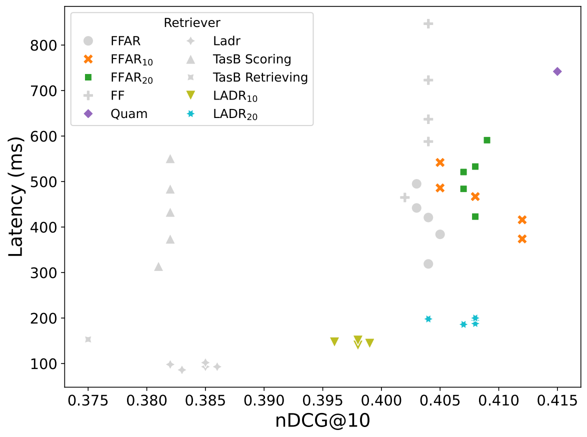
Next, our additional cross-encoder stage for LADR and FFAR adds relatively low latency, while almost always exceeding effectiveness. This shows promise that even a small cross-encoder budget can improve effectiveness notably. However, our cross-encoder stage means that our solution needs to run on a GPU instead of a CPU, making it more expensive regarding resource usage. Moreover, LADR generally seems to gain more from using cross-encoders than FFAR.

Retriever	Device	TREC DL Hard				
		nDCG@10	*RR*	*AP@10*	*AP@c*	*Latency (ms)*
BM25	***CPU***	0.274	0.422	0.094	0.148	-
Fast Forward	***CPU***	0.402	0.544	0.169	**0.25**	465
TasB Retrieval	***CPU***	0.375	0.512	0.159	0.236	153
TasB Scoring	***CPU***	0.381	0.512	0.159	0.241	314
Quam	*GPU*	**0.415**	**0.57**	**0.173**	0.243	742
LADR	***CPU***	0.383	0.514	0.162	0.238	**86**
LADR_{10} (MonoT5)	*GPU*	0.398	0.523	0.165	-	140
LADR_{20} (MonoT5)	*GPU*	0.407	0.54	0.166	-	186
FFAR	***CPU***	0.404	0.543	0.169	0.246	319
FFAR_{10} (MonoT5)	*GPU*	0.412	0.541	0.168	-	374
FFAR_{20} (MonoT5)	*GPU*	0.408	0.55	0.162	-	423

Table 5.7: Retrieval performance in effectiveness and efficiency on (TREC DL Hard) for retrieval budget $c = 8192$.



(a) CPU-based systems highlighted.



(b) GPU-based systems highlighted.

Figure 5.9: Comparison of effectiveness against efficiency on (TREC DL Hard). Different data points for the same retrieval system are for different budgets.

Overall, we can see that the cross-encoder-based solutions perform better in effectiveness than their non-cross-encoder-based counterparts. This effect seems more pronounced in the TREC DL Hard dataset. The more challenging nature of the queries in this dataset likely makes them benefit more from the additional signals that a cross-encoder can extract by directly comparing the queries to the documents.

Cross-encoders

The results of this experiment can be found in Table 5.8. We run experiments for three different cross-encoder budgets, corresponding to budget configurations used in the Quam paper [34]. On a budget of 50 cross-encoder calls, we see that FFAR has the highest effectiveness on both datasets, while Quam scores the lowest on the 2019 dataset and LADR scores the lowest on the 2020 dataset. We see that Quam shows the highest effectiveness for higher budgets. LADR is the most efficient solution in every case, with Quam second on a budget of 50 and 100 cross-encoder calls. We see that Quam is the least efficient solution on a budget of 250 calls.

CE Budget	**Retriever**	TREC DL 2019		TREC DL 2020	
		nDCG@10	*Latency (ms)*	*nDCG@10*	*Latency (ms)*
50	FFAR	**0.727**	463	**0.715**	602
	Quam	0.697	360	0.714	440
	LADR	0.723	**286**	0.705	**367**
100	FFAR	0.722	700	0.702	912
	Quam	**0.724**	679	**0.718**	847
	LADR	0.718	**526**	0.696	**686**
250	FFAR	0.715	1502	0.695	1935
	Quam	**0.729**	1721	**0.714**	2170
	LADR	0.714	**1333**	0.693	**1737**

Table 5.8: Retrieval performance in effectiveness and efficiency for cross-encoder-based systems. FFAR and LADR use an initial budget of $c = 8192$.

We see that effectiveness for FFAR and LADR decreases as our cross-encoder budget increases. This suggests that our cross-encoder is less effective than our dense re-ranker. For FFAR, another possibility exists for this decline in effectiveness. Although we apply score interpolation between lexical and dense scores, this interpolated score is not used during the cross-encoder re-ranking stage. An additional interpolation step that includes the cross-encoder score could potentially improve effectiveness.

Another notable observation is that Quam's latency increase is steeper than the other retrieval systems. While FFAR and LADR introduce all additional overhead at the beginning of the retrieval pipeline, Quam introduces the additional overhead during its re-ranking stage. This extra overhead causes Quam's re-ranking stage to be slower and likely leads to a steeper increase in latency compared to the other retrievers as the budget grows.

6 Conclusions and Future Work

6.1. Conclusions

This research investigates the trade-off between efficiency and effectiveness in adaptive retrieval for efficient retrieval systems. Adaptive retrieval is developed to improve the effectiveness of retrieval systems by leveraging the relationships between highly ranked documents and their neighbours. Its efficient graph-based design makes adaptive retrieval a promising approach for improving efficient retrieval systems.

RQ1: How can a corpus graph be used most effectively to overcome the limitations of first-stage lexical retrievers?
Given that the corpus graph is central to adaptive retrieval, we aim to better understand how to leverage it most effectively. We propose a visualisation tool called `GarVis` to intuitively explore the structure of neighbourhoods constructed using a corpus graph. We observe that adaptive retrieval has mixed effects across queries, with some showing improvements and others experiencing a decline. Moreover, we find examples of document relationships that indicate that the clustering hypothesis holds to a certain extent.

Next, we present an approach for automatically analysing neighbourhood configurations across various retrieval budgets. Our experiments identify configurations for different budgets that use the corpus graph most effectively to improve candidate set recall (see Table 5.2). Additionally, we find that using adaptive retrieval introduces duplicate documents to the dataset, reducing the effective size of the candidate set by up to 40% of the original size. We propose removing duplicate documents as an efficient strategy for handling duplicate documents in the candidate set.

RQ2: How do scoring mechanisms influence the effectiveness of adaptive retrieval?
Adaptive retrieval does not directly affect relevance scores but aims to increase effectiveness by extending the candidate pool. We experiment with how different scoring mechanisms interact with adaptive retrieval. We investigate the effect of interpolation-based re-ranking, which combines scores from lexical and semantic retrievers, to understand its impact on adaptive retrieval. Although interpolation relies on lexical scores, which may hurt adaptive retrieval because newly found documents often have lower lexical scores, we find that interpolation has a positive effect. This indicates that score interpolation functions as intended, effectively balancing lexical and semantic signals. Moreover, we experiment with an additional cross-encoder stage in the efficient adaptive retrieval pipeline and find that we can achieve notable improvements in effectiveness with a small budget of cross-encoder calls.

RQ3: How does adaptive retrieval affect efficient re-ranking in effectiveness and efficiency?
In this research, the trade-off between effectiveness and efficiency is central. We compare `FFAR` against various baselines. We compare `FFAR` against Fast Forward, its counterpart without adaptive retrieval, and find that `FFAR` is less effective but more efficient. This result is curious, as adaptive retrieval typically aims to improve effectiveness at the cost of additional overhead. However, this can be explained by

the construction of the candidate set, which includes many duplicate documents that are removed. As a result, the re-ranking phase becomes less expensive, but at the cost of effectiveness.

Additionally, we compare FFAR with LADR and its cross-encoder variants and find that the FFAR variants are generally more effective and LADR variants are more efficient. The higher effectiveness of FFAR is expected, as we demonstrate that score interpolation positively impacts effectiveness. Similarly, we expect LADR to be more efficient, given that FFAR introduces extra overhead due to score interpolation and lexical score computation. However, we observe a much greater latency difference between LADR and FFAR than anticipated, which we attribute to differences in implementation. Overall, LADR_{20} seems to strike the best balance between efficiency and effectiveness.

Overall, we find that FFAR_{20} achieves the highest effectiveness on two datasets, while Quam is the most effective on the Hard dataset. Cross-encoder-based solutions generally outperform their non-cross-encoder counterparts in effectiveness, which seems more pronounced on the Hard dataset. This is likely due to the dataset's more challenging nature, which makes the direct query-to-document comparisons provided by cross-encoders more valuable.

Finally, we compare Quam more closely with our efficient adaptive retrieval pipelines with an additional cross-encoder stage. We find efficient adaptive retrieval solutions are more effective when using a small cross-encoder budget. However, as the budget increases, Quam becomes the most effective solution, although with a steeper increase in latency.

6.2. Future Work and Limitations

Adaptive retrieval is a technique focused on increasing the recall of the candidate set. Increased recall does not necessarily mean that newly found relevant documents are ranked highly by the scorer, as adaptive retrieval has no direct influence on document scoring. While the corpus graph provides similarity scores between documents, these signals are typically ignored during re-ranking. An interesting research direction would be to explore how these similarity scores might be integrated into the document scoring process. Hybrid re-ranking methods have demonstrated the benefits of incorporating lexical signals into final relevance scores. A similar approach to integrating the document similarity scores could improve effectiveness in adaptive retrieval settings. One possible direction is to apply (weighted) reciprocal rank fusion based on the degree of neighbouring documents. Alternatively, interpolating the similarity scores offers another approach to incorporating document similarity into the score. Further experiments are needed to evaluate whether document similarity can provide meaningful additional signals for scoring.

Quam introduces the concept of a Learnt Affinity graph, which is created by recomputing edge weights using a Query Affinity Model (see Section 4.4). This graph leads to significant improvements in effectiveness, yet we do not incorporate this approach in our paper. While Quam provides a pre-computed Laff graph, it is unsuitable for our experiments, as it contains only 128 edges and is constructed from the BM25 graph. Since our experiments involve up to 127 neighbours, the recomputed edge weights would offer minimal benefit. Additionally, we prefer not to use a BM25 graph as the base, opting instead for TasB. This means that to incorporate the Laff graph into our experiments, we need to compute our Laff graph. We decide that computing the graph for Tasb is too expensive for these experiments, requiring re-scoring every single edge. Nonetheless, it is an interesting research direction to see how a Laff graph impacts the efficient adaptive retrieval pipeline.

We find that LADR is faster than FFAR, likely due to differences in their implementations. Due to time constraints, we can not incorporate score interpolation into LADR to explore its efficiency-effectiveness trade-off. An interesting direction for future research can involve efficiently integrating score interpolation into LADR, potentially resulting in a more efficient variant of FFAR.

Finally, in our research involving an additional cross-encoder stage, we perform score interpolation between sparse and dense scores before passing them to the cross-encoder. However, the cross-encoder discards this score. Interpolating the cross-encoder score with previous scores could improve effectiveness, presenting another direction for future research.

6.3. Acknowledgements

Following TU Delft policies[1], we acknowledge the use of AI tools in this work. ChatGPT and Grammarly were used to improve sentence structure, grammar, and reduce repetition for a more polished delivery.

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

References

[1] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. 11 2016. URL `http://arxiv.org/abs/1611.09268`.

[2] Sebastian Bruch, Siyu Gai, and Amir Ingber. An Analysis of Fusion Functions for Hybrid Retrieval. *ACM Transactions on Information Systems*, 42(1), 8 2023. ISSN 15582868. doi: 10.1145/3596512.

[3] Sebastian Bruch, Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. Seismic: Efficient and Effective Retrieval over Learned Sparse Representation. Technical report, 2024. URL `https://big-ann-benchmarks.com/neurips23.html`.

[4] Eunseong Choi, Sunkyung Lee, Minijn Choi, Hyeseon Ko, Young In Song, and Jongwuk Lee. SpaDE: Improving Sparse Representations using a Dual Document Encoder for First-stage Retrieval. In *International Conference on Information and Knowledge Management, Proceedings*, pages 272–282. Association for Computing Machinery, 10 2022. ISBN 9781450392365. doi: 10.1145/3511808.3557456.

[5] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. OVERVIEW OF THE TREC 2020 DEEP LEARNING TRACK. Technical report, . URL `https://microsoft.github.io/TREC-2020-Deep-Learning`.

[6] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. OVERVIEW OF THE TREC 2019 DEEP LEARNING TRACK. Technical report, .

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova Google, and A I Language. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Technical report. URL `https://github.com/tensorflow/tensor2tensor`.

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

[9] Meet Doshi, Vishwajeet Kumar, Rudra Murthy, Vignesh P, and Jaydeep Sen. Mistral-SPLADE: LLMs for better Learned Sparse Retrieval. 8 2024. URL `http://arxiv.org/abs/2408.11119`.

[10] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The Faiss library. 1 2024. URL `http://arxiv.org/abs/2401.08281`.

[11] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *SIGIR 2021 - Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2288–2292. Association for Computing Machinery, Inc, 7 2021. ISBN 9781450380379. doi: 10.1145/3404835.3463098.

[12] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. Technical report, 2020.

[13] Sebastian Hofstätter, Sheng Chieh Lin, Jheng Hong Yang, Jimmy Lin, and Allan Hanbury. Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling. In *SIGIR 2021 - Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 113–122. Association for Computing Machinery, Inc, 7 2021. ISBN 9781450380379. doi: 10.1145/3404835.3462891.

[14] Thomas Jaenich, Graham McDonald, and Iadh Ounis. Fairness-Aware Exposure Allocation via Adaptive Reranking. pages 1504–1513. Association for Computing Machinery (ACM), 2024. doi: 10.1145/3626772.3657794.

[15] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. ISSN 01628828. doi: 10.1109/TPAMI.2010.57.

[16] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. 2 2017. URL `http://arxiv.org/abs/1702.08734`.

[17] Hrishikesh Kulkarni, Sean MacAvaney, Nazli Goharian, and Ophir Frieder. Lexically-Accelerated Dense Retrieval. In *SIGIR 2023 - Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 152–162. Association for Computing Machinery, Inc, 2023. doi: 10.1145/3539618.3591715.

[18] Jurek Leonhardt, Henrik Müller, Koustav Rudra, Megha Khosla, Abhijit Anand, and Avishek Anand. Efficient Neural Ranking Using Forward Indexes and Lightweight Encoders. *ACM Transactions on Information Systems*, 42(5), 2024. ISSN 15582868. doi: 10.1145/3631939.

[19] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. Distilling Dense Representations for Ranking using Tightly-Coupled Teachers. 10 2020. URL `http://arxiv.org/abs/2010.11386`.

[20] Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and David R Cheriton. In-Batch Negatives for Knowledge Distillation with Tightly-Coupled Teachers for Dense Retrieval. Technical report, 2021.

[21] Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. Fine-Tuning LLaMA for Multi-Stage Text Retrieval. In *SIGIR 2024 - Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2421–2425. Association for Computing Machinery, Inc, 7 2024. ISBN 9798400704314. doi: 10.1145/3626772.3657951.

[22] Sean MacAvaney, Nicola Tonellotto, and Craig MacDonald. Adaptive Re-Ranking with a Corpus Graph. In *International Conference on Information and Knowledge Management, Proceedings*, pages 1491–1500. Association for Computing Machinery, 2022. doi: 10.1145/3511808.3557231.

[23] Joel Mackenzie. Approximate Bag-of-Words Top-k Corpus Graphs. Technical report, 2025.

[24] Iain Mackie, Jeffery Dalton, and Andrew Yates. How Deep is your Learning: the DL-HARD Annotated Deep Learning Dataset. 5 2021. URL `http://arxiv.org/abs/2105.07975`.

[25] Yu A Malkov and D A Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. Technical report.

[26] Thong Nguyen, Shubham Chatterjee, Sean MacAvaney, Iain Mackie, Jeff Dalton, and Andrew Yates. DyVo: Dynamic Vocabularies for Learned Sparse Retrieval with Entities. 10 2024. URL `http://arxiv.org/abs/2410.07722`.

[27] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. Multi-Stage Document Ranking with BERT. 10 2019. URL `http://arxiv.org/abs/1910.14424`.

[28] Alec Radford Openai, Karthik Narasimhan Openai, Tim Salimans Openai, and Ilya Sutskever Openai. Improving Language Understanding by Generative Pre-Training. Technical report. URL `https://gluebenchmark.com/leaderboard`.

[29] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data. *Proceedings of the ACM on Management of Data*, 2(3):1–27, 5 2024. doi: 10.1145/3654923.

[30] Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models. 1 2021. URL `http://arxiv.org/abs/2101.05667`.

[31] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. RankVicuna: Zero-Shot Listwise Document Reranking with Open-Source Large Language Models. 9 2023. URL http://arxiv.org/abs/2309.15088.

[32] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. RankZephyr: Effective and Robust Zero-Shot Listwise Reranking is a Breeze! 12 2023. URL http://arxiv.org/abs/2312.02724.

[33] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Technical report, 2020. URL http://jmlr.org/papers/v21/20-074.html.

[34] Mandeep Rathee, Sean MacAvaney, and Anand Avishek. Quam: Adaptive Retrieval through Query Affinity Modelling. In *IEEE International Conference on Program Comprehension*, volume 2022-March, pages 36–47. IEEE Computer Society, 2024. ISBN 9781450392983. doi: 10.1145/nnnnnnn.nnnnnnn.

[35] Mandeep Rathee, Sean MacAvaney, and Avishek Anand. Guiding Retrieval using LLM-based Listwise Rankers. 1 2025. URL http://arxiv.org/abs/2501.09186.

[36] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. ISSN 15540669. doi: 10.1561/1500000019.

[37] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, and others. Okapi at TREC-3. *Nist Special Publication Sp*, 109:109, 1995.

[38] Guilherme Rosa, Luiz Bonifacio, Vitor Jeronymo, Hugo Abonizio, Marzieh Fadaee, Roberto Lotufo, and Rodrigo Nogueira. In Defense of Cross-Encoders for Zero-Shot Retrieval. 12 2022. URL http://arxiv.org/abs/2212.06121.

[39] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. Introduction to information retrieval. 39, 2008.

[40] K Sparck Jones, S Walker, and S E Robertson. A probabilistic model of information retrieval: development and comparative experiments Part 2. Technical report. URL www.elsevier.com/locate/infoproman.

[41] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. 4 2023. URL http://arxiv.org/abs/2304.09542.

[42] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models. 4 2021. URL http://arxiv.org/abs/2104.08663.

[43] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[44] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk Microsoft. APPROXIMATE NEAREST NEIGHBOR NEGATIVE CON-TRASTIVE LEARNING FOR DENSE TEXT RETRIEVAL. Technical report. URL https://aka.ms/ance.

[45] Yingrui Yang, Parker Carlson, Shanxiu He, Yifan Qiao, and Tao Yang. Cluster-based Partial Dense Retrieval Fused with Sparse Text Retrieval. In *SIGIR 2024 - Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2327–2331. Association for Computing Machinery, Inc, 7 2024. ISBN 9798400704314. doi: 10.1145/3626772.3657972.

[46] Haoyu Zhang, Jun Liu, Zhenhua Zhu, Shulin Zeng, Maojia Sheng, Tao Yang, Guohao Dai, and Yu Wang. Efficient and Effective Retrieval of Dense-Sparse Hybrid Vectors using Graph-based Approximate Nearest Neighbor Search. 10 2024. URL http://arxiv.org/abs/2410.20381.