

Building a Robust Retrieval System with Dense Retrieval Models

by

Sheng-Chieh Lin

A thesis
presented to the University Of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2024

© Sheng-Cheih Lin 2024

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Member: Siva Reddy
Assistant Professor
School of Computer Science
McGill University

Supervisor(s): Jimmy Lin
Professor
David R. Cheriton School of Computer Science
University of Waterloo

Internal Member: Charles Clarke
Professor
David R. Cheriton School of Computer Science
University of Waterloo

Internal Member: Victor Zhong
Assistant Professor
David R. Cheriton School of Computer Science
University of Waterloo

Internal-External Member: Elliot Creager
Assistant Professor
Department of Electrical and Computer Engineering
University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The thesis includes contributions from five published papers. The thesis author Sheng-Chieh Lin is the first author of all the five papers, who is responsible for the most parts of ideation, methodology, implementation, conducting experiments, analyzing results, and paper writing. List of publications and co-authors:

- Chapter 3: In-Batch Negatives for Knowledge Distillation with Tightly-Coupled Teachers for Dense Retrieval. Sheng-Chieh Lin*, Jheng-Hong Yang*, Jimmy Lin. ACL workshop on Representation Learning for NLP, August 2021.
- Chapter 4: How to Train Your DRAGON: Diverse Augmentation Towards Generalizable Dense Retrieval. Sheng-Chieh Lin*, Akari Asai, Minghan Li, Barlas Oguz, Jimmy Lin, Yashar Mehdad, Wen-tau Yih, Xilun Chen*. In Proceedings of the Conference on Empirical Methods in Natural Language Processing Findings, Dec 2023.
- Chapter 5: A Dense Representation Framework for Lexical and Semantic Matching. Sheng-Chieh Lin, Jimmy Lin. ACM Transactions on Information Systems, April 2023.
- Chapter 6: Aggretriever: A Simple Approach to Aggregate Textual Representation for Robust Dense Passage Retrieval. Sheng-Chieh Lin, Minghan Li, Jimmy Lin. Transactions of the Association for Computational Linguistics, May 2023.
- Chapter 7: mAggretriever: A Simple yet Effective Approach to Zero-Shot Multilingual Dense Retrieval. Sheng-Chieh Lin, Amin Ahmad, Jimmy Lin. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, Dec 2023.

Abstract

First-stage retrieval plays a pivotal role in modern search engines, aiming to gather candidates from a large corpus for various downstream tasks, such as question answering, fact-checking, and retrieval-augmented generation. Historically, retrieval using heuristic lexical representations, such as BM25, has been widely employed as a first-stage retriever. Recently, driven by advancements in pre-trained transformers and approximate nearest neighbor search, transformer-based dense retrieval models have emerged as a strong alternative to BM25. These models address the issue of term mismatch in lexical representations by enabling semantic matching.

Unlike BM25, transformer-based dense retrieval models require substantial resources for training, such as human relevance labels and GPUs. Moreover, even with sufficient training data, these models often lag behind BM25 in robustness across different retrieval domains, tasks, and languages, limiting their broader applicability. As an alternative, dense-sparse fusion and multi-vector bi-encoder retrieval models have been shown to capture both semantic and lexical features, demonstrating superior retrieval robustness. However, these designs introduce increased query latency and require a more complex pipeline for indexing and retrieval, making them less suitable for first-stage retrieval systems. To address these challenges and enable a robust yet efficient first-stage retrieval system, this thesis contributes to the field of information retrieval in two key directions. First, we propose efficient training methods to enhance the effectiveness of transformer-based dense retrieval models. Second, we introduce a dense representation framework that simplifies the integration of dense and sparse (semantic and lexical matching) indexing and retrieval.

To begin, we propose a method to enhance the efficiency of a widely adopted training technique for dense retrieval models: knowledge distillation. Rather than distilling knowledge from a more powerful cross-encoder ranker, we leverage a multi-vector bi-encoder model (i.e., ColBERT) to train a dense retrieval model, which we call TCT-ColBERT. Specifically, we transfer knowledge from the bi-encoder teacher to the student by distilling ColBERT’s fine-grained scoring function into a simpler dot-product operation. The bi-encoder teacher-student setup offers a significant advantage: it allows for the efficient incorporation of in-batch negatives during knowledge distillation, facilitating richer interactions between the teacher and student models. Our results demonstrate that this approach improves both the training efficiency and the overall effectiveness of dense retrieval models compared to prior methods.

Despite the success of dense retrieval models, we observe that they still fall short of multi-vector dense retrieval models and even BM25 in terms of robustness across various retrieval domains and tasks. Recent studies have explored scaling dense retrieval models by increasing both training data and model size. However, this approach necessitates collecting billions of

training samples from the web, significantly driving up training costs. To enhance the efficiency of training robust dense retrieval models, we systematically investigate dense retrieval training through the lens of data augmentation and identify a critical factor: diverse query and label augmentation. Consequently, we are the first to empirically demonstrate that a BERT-base-sized dense retrieval model (110M parameters), named DRAGON, can rival the state-of-the-art dense retrieval model, GTR-XXL, with 4.8 billion parameters (40 times larger) in both supervised and zero-shot evaluations. Crucially, instead of relying on billion-scale web-crawled data, our data augmentation approach uses only the 8.8 million passages from the MS MARCO corpus.

Subsequently, we explore deploying a more effective hybrid retrieval model efficiently. To achieve this, we propose a dense representation framework for semantic and lexical matching, incorporating a novel textual representation and scoring function. This unified framework simplifies the fusion retrieval of any off-the-shelf dense (semantic) and sparse (lexical) retrievers with a single index. In addition, the fusion retrieval can be accelerated using GPUs and efficient approximate nearest neighbor search through existing libraries. Moreover, this framework enables the development of a newly designed bi-encoder hybrid retrieval model, (DeLADE+[CLS])_{DHR}, which achieves competitive retrieval effectiveness while significantly reducing runtime costs compared to existing bi-encoder multi-vector retrievers.

Finally, we introduce a new dense retrieval model, Aggretriever. In contrast to existing dense retrieval models, which derive semantic textual representations solely from [CLS] tokens or average pooling, Aggretriever fully leverages the capabilities of pre-trained transformers by integrating semantic features from [CLS] and lexical features from masked language modeling. Our experiments highlight the advantages of Aggretriever over existing dense retrieval models in terms of training efficiency and robustness. Specifically, Aggretriever achieves competitive retrieval effectiveness in both supervised and zero-shot evaluations while requiring only a single V100 GPU (32GB) and avoiding complex, costly training strategies such as knowledge distillation and hard negative mining. Additionally, we extend Aggretriever to a multilingual dense retrieval model, mAggretriever, demonstrating its superior zero-shot transferability across languages. On multilingual benchmarks, mAggretriever, fine-tuned solely on English training data, outperforms existing multilingual dense retrieval models that rely on computationally expensive pre-training with large-scale multilingual datasets crawled from the web.

In conclusion, we summarize our findings and explore potential future research directions. For instance, we highlight the promising opportunities of leveraging large language models (LLMs) to develop more robust retrieval systems. These systems could enable advanced capabilities such as search planning and multimodal retrieval, paving the way for more versatile and effective information retrieval frameworks.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Jimmy Lin, for his unwavering guidance and support throughout my PhD journey. From mentoring me on my very first project on information retrieval in TREC to teaching me how to effectively present and structure my research ideas, his insights have been truly invaluable. I am particularly thankful for the trust he placed in me, empowering me to manage my own projects and granting me the freedom to explore my research interests independently.

I would like to extend my heartfelt thanks to my committee members, Charles, Victor, Elliot, and Siva, for taking the time to thoroughly review my lengthy thesis and for providing invaluable suggestions that significantly improved the quality of my work.

I am deeply grateful to my mentors and colleagues from my internships at Meta, Vectara, and Nvidia, including Xilun, Akari, Luyu, Barlas, Wenhan, Scott, Amin, Chankyu, and Wei. Your guidance has taught me how to identify impactful research problems, think critically, and bridge the gap between research and industry. Your support has been instrumental in shaping both my academic journey and professional development.

I would like to express my heartfelt thanks to all my friends from the DSG lab in Waterloo. A special thanks to Matt, for being both my roommate and co-author on many papers. Our collaborations have been invaluable in helping me enhance my research and paper-writing skills. I am also deeply grateful to Minghan, Xueguang, and Wei, for being my co-authors and for offering encouragement during challenging times. More importantly, we shared enjoyable moments learning how to snowboard and discussing fascinating research ideas. I would also like to thank Rodrigo and Ronak for making TREC such an enjoyable experience, and Zhiying, Crystina, Jayden, Yuqing, and Nandan for the memorable time we spent together in the lab.

I would like to thank my advisors, Chuan-Ju and Ming-Feng, from Academia Sinica, where I worked as a research assistant before starting my PhD journey. Thank you for your patience and guidance during my first machine learning research project after I made the transition from the semiconductor field to computer science. I am also grateful to my colleagues Chia-Yuan, Yu-Neng, Ting-Hsiang, and Ting-Wei for the memorable time we spent together at Academia Sinica. A special thanks to my friends Henry, Cheng-Hao, and Kevin, for the support and camaraderie as we prepared for career changes and graduate studies.

Finally, I would like to extend my heartfelt thanks to my family for their unwavering support throughout this journey. Their encouragement, especially during my transition from working in the industry to pursuing a PhD, has been invaluable. Our weekend chats have provided relaxation and comfort, helping me navigate through the most stressful times. I am also deeply grateful to my brother and Aimee for helping me settle in during my first internship in Seattle. The time we

spent together, visiting beautiful places, remains one of the most memorable parts of my PhD journey.

Dedication

To my parents, my brother and all my friends.

Table of Contents

List of Figures	xiii
List of Tables	xvi
1 Introduction	1
1.1 Challenges of Deploying a Robust Dense Retrieval Model	3
1.2 Our Proposed Solutions for Challenges	4
1.3 Contributions and Outline	5
2 Background	7
2.1 Ad-Hoc Text Retrieval and Ranking Models	7
2.2 Bi-Encoder Retrieval Models	9
2.3 Training Bi-Encoder Retrieval Models	11
2.4 Indexing and Retrieval	12
2.5 Datasets and Evaluation Metrics for Text Retrieval	14
2.6 Related Work	15
3 Efficiently Training a Dense Retrieval Model	17
3.1 Overview	17
3.2 Methodology: TCT-ColBERT	18
3.3 Experimental Setups	21
3.4 Experimental Results	22
3.5 Summary	27

4	Training a Robust Dense Retrieval Model with Diverse Augmentation	28
4.1	Overview	28
4.2	A Unified Framework of Improved Dense Retrieval Training: Data Augmentation	30
4.3	Pilot Studies on Data Augmentation	31
4.4	Methodology: DRAGON	36
4.5	Experimental Setups	37
4.6	Experimental Results	40
4.7	Summary	46
5	A Dense Representation Framework for Semantic and Lexical Matching	47
5.1	Overview	47
5.2	Methodology	49
5.3	Experimental Setups	56
5.4	Experimental Results	59
5.5	Summary	76
6	Aggregating Semantic and Lexical Features for Robust Dense Retrieval	77
6.1	Overview	77
6.2	Methodology: Aggretriever	79
6.3	Experimental Setups	83
6.4	Experimental Results	85
6.5	Summary	93
7	A Simple Yet Effective Approach to Zero-Shot Multilingual Dense Retrieval	94
7.1	Overview	94
7.2	Methodology: mAggretriever	95
7.3	Experimental Setups	97
7.4	Experimental Results	99
7.5	Summary	102

8 Conclusion and Future Work	103
References	105

List of Figures

1.1	An illustration of a first-stage retrieval model.	2
2.1	Three main architectures of ranking models.	8
3.1	Illustration of the differences between pairwise knowledge distillation (KD) and our proposed in-batch knowledge distillation.	19
3.2	Passage retrieval effectiveness on a synthetic corpus comprising relevant passages and BM25 results as additional “distractors” randomly sampled from the corpus are added.	27
4.1	Supervised versus zero-shot effectiveness comparison among existing state-of-the-art retrievers. All models use a BERT-base-sized (110M parameters) backbone except for GTR-XXL (4.8B parameters).	29
4.2	Illustration of progressive label augmentation. For each iteration of training, additional relevance labels from a teacher are augmented in the training data. By contrast, uniform supervision directly exposes models to all the supervisions (as in iteration 3) in the beginning.	34
4.3	Impacts of query augmentation.	37
4.4	Examples of augmented queries and relevance labels from a passage. Augmented rel (#) denotes the number of unique relevant passages labeled by all our five teachers.	45

5.1	Illustration of densified lexical representations (DLRs) and gated inner product (GIP). Both query and passage lexical representations are fixed-width vectors where the number of dimensions is equal to the vocabulary size. Our approach first groups these high-dimensional vectors into M slices, each with N dimensions (e.g., $M = 3$, $N = 5$ here). For each slice, the maximum value is selected. These values from the original vector (Value) and their positions in each slice (Index) are recorded separately. When computing the gated inner product between two vectors, only the dimensions with the same index are considered.	52
5.2	Approximate GIP. DLR retrieval can be approximated by only computing the gated inner product from a few dimensions where the query value vector has values above a threshold (e.g., greater than 0.5 here). In two-stage retrieval, we can then precisely rerank the top- K passages using all the dimensions.	53
5.3	Recall comparison of approximate retrieval approaches at different cutoffs. Approx. GIP refers to approximate gated inner product with threshold θ . Inner product refers to standard inner product between query and passage value vectors without involving the index vectors.	67
5.4	Two-stage retrieval performance comparisons. We compare retrieval latency and effectiveness between approximate gated inner product (Approx. GIP) and standard inner product retrieval (between query and passage value vectors without involving the index vectors) and their performance after GIP reranking the top-10000 retrieved candidates. The labeled numbers denote the threshold θ for Approx. GIP retrieval. Note that approx. GIP with $\theta = 0$ is equal to GIP retrieval without any approximation.	68
6.1	In-domain vs zero-shot effectiveness. All models are trained with the negatives mined by BM25.	78
6.2	(a) BERT: next sentence prediction (NSP) and mask language modeling (MLM) (b) DPR: using the [CLS] embedding for retrieval (c) Aggretriever: aggregating knowledge from both NSP and MLM.	79
6.3	Illustration of text aggregation: (a) pooling of token representations to form $\mathbf{v}_q$; (b) pruning of $\mathbf{v}_q$ to form agg_q^* (or agg_q^+). While pruning, $\text{agg}_q^*[n]$ receives a negative value if the pooled element belongs to S_n^- ; i.e., the second element in each slice (red box).	81

- 7.1 Illustration of MLM prediction for the input token q_1 . (1) Whole-vocabulary prediction considers all languages while target-language prediction only considers the tokens in one language; e.g., (2) source language (Spanish) or (3) English. (4) Self prediction assigns probability of one to itself. 0. denotes less than 0.1. . . 96

List of Tables

3.1	Training cost comparison. We report the training time per batch against the baseline (without a teacher model) on a single TPU-v2. Our backbone model is BERT-base, with batch size 96. The in-batch cross-encoder training time is not available because it exceeds the memory limit.	20
3.2	Passage retrieval results with BM25 negative training. For knowledge distillation (KD) methods, the effectiveness of teacher (T) models is also reported. All our implemented models are labeled with a number and superscripts represent significant improvements over the labeled model (paired t -test, $p < 0.05$). Note that Bonferroni correction is not applied, so the significance tests may risk overconfidence.	23
3.3	Passage retrieval results with hard negative training. All our implemented models are labeled with a number and superscripts represent significant improvements over the labeled model (paired t -test, $p < 0.05$). Note that Bonferroni correction is not applied, so the significance tests may risk overconfidence.	24
3.4	Document retrieval results using the FirstP approach. All our implemented models are labeled with a number and superscripts represent significant improvements over the labeled model (paired t -test, $p < 0.05$). Note that Bonferroni correction is not applied, so the significance tests may risk overconfidence.	26
4.1	Categorization of existing dense retrieval models by their approaches to data augmentation.	30
4.2	Strategies to obtain multiple supervisions using cropped sentences as queries. . .	35
4.3	Study on trajectory of progressive supervision using cropped sentences as queries.	36
4.4	MS MARCO and our augmented training queries statistics.	38
4.5	The latency comparison of relevance label augmentation with batch inference using different teachers on MS MARCO.	39

4.6	Comparison with existing state-of-the-art dense retrievers. Bold (underline) denotes the best (second best) effectiveness for each row among baseline dense models.	41
4.7	Label augmentation with cross encoder (CE) using cropped sentences as queries.	42
4.8	Ablation on initialization using the mixture of cropped sentences and GenQ as queries.	43
4.9	Ablation on progressive label augmentation from top- k passages using cropped sentences as queries with trajectory: uniCOIL $\rightarrow$ Contriever $\rightarrow$ ColBERTv2.	44
4.10	Uniform and progressive supervision effectiveness comparison at each training iteration. The models are trained using cropped sentences as queries.	44
5.1	Comparison of single-vector representation learning approaches.	49
5.2	Effectiveness/Efficiency Comparisons of Independent Fusion of DLRs with “Off-the-Shelf” Dense Semantic Representations on MS MARCO (Dev)	59
5.3	Effectiveness/efficiency comparisons. This table compares against a selection of retrieval models trained with comparable baseline strategies. Bold (underline) denotes the best (second best) effectiveness for each row. For the MS MARCO datasets, superscripts denote significant improvements over the labeled model with paired t -test ($p < 0.05$). Note that Bonferroni correction is not applied, so the significance tests may risk overconfidence.	60
5.4	Component retrieval effectiveness of (DeLADE+[CLS]) _{DHR} (768)	61
5.5	Ablation of (DeLADE+[CLS]) _{DHR} (768) varying the semantic component. Bold denotes the best effectiveness for each column. Superscript denotes significant improvement over the labeled model based on paired t -tests ($p < 0.05$). Note that Bonferroni correction is not applied, so the significance tests may risk overconfidence.	61
5.6	Effectiveness/efficiency comparisons with existing state-of-the-art models using advanced training techniques. Bold (underline) denotes the best (second best) effectiveness for each row.	64
5.7	Effects of θ on Approximate GIP with the Reconstructed Expanded Query from DeLADE _{DLR} (768)	69

5.8	End-to-end retrieval latency for (DeLADe+[CLS]) _{DHR} (768) on MS MARCO (Dev). Retrieval latency is measured in terms of ms/q with a single thread and batch size 1. With the exception of row (1), for each query, we retrieve the top-10000 passages at the first stage and then rerank the candidates using the second stage. Approx. GIP refers to approximate gated inner product with threshold $\theta = 0.3$. Inner product refers to standard inner product between query and passage value vectors without involving the index vectors.	70
5.9	Comparison of Vector Densification Settings for the MS MARCO Passage Corpus	71
5.10	Effectiveness/efficiency comparisons of DLRs with different base models on MS MARCO (Dev).	73
5.11	Comparison of Different DLR Slicing Strategies on MS MARCO (Dev).	74
5.12	Effectiveness comparisons of DLRs with different unsupervised compression techniques on FiQA-2018 (Test).	75
6.1	Supervised retrieval effectiveness comparisons. All models are fine-tuned with negatives from BM25. Bold denotes the best model for that metric.	85
6.2	Near-domain zero-shot retrieval effectiveness comparisons using NQ or TQA for fine-tuning. Bold denotes the best model for that metric.	86
6.3	Multi-domain zero-shot retrieval effectiveness comparisons using various sources for fine-tuning. Bold denotes the best model for that metric.	87
6.4	Comparisons with state-of-the-art dense retrieval models.	88
6.5	Fine-tuning with limited data.	89
6.6	Fine-tuning with noisy hard negatives.	90
6.7	DistilBERT _{AGG} text aggregation ablation. We project [CLS] to 128 dimensions and concatenate with a 640-dimensional embedding pooled and pruned using different strategies. AVERAGE denotes average pooling over all 768-dimensional contextualized token embeddings other than [CLS].	91
6.8	DistilBERT _{AGG} dimensionality ablation.	92
6.9	Query encoding latency comparisons.	93
7.1	Training and inference cost comparisons on MS MARCO with XLM-R base. . .	97
7.2	MIRACL and XQuAD-R data statistics.	98

7.3	Supervised (MARCO Dev) English and zero-shot (MIRACL Dev) multilingual retrieval effectiveness comparisons. The best effectiveness in each column is highlighted in bold.	100
7.4	Zero-shot retrieval effectiveness on XQuAD-R.	101
7.5	Mr. TyDi multilingual retrieval comparisons with state-of-the-art multilingual retrievers. Bold denotes the best effectiveness. multi CL denotes multilingual contrastive learning.	102

Chapter 1

Introduction

First-stage text retrieval is a crucial component for many downstream tasks, such as question answering [Karpukhin et al., 2020], fact-checking [Wachsmuth et al., 2018b], retrieval-augmented generation [Lewis et al., 2020, Li et al., 2024], and alignment for large language models [Lin et al., 2024]. These text retrieval systems help select a few relevant candidates for subsequent processing from a corpus containing millions, or even trillions, of candidates. In a first-stage retrieval system, both queries and passages are encoded independently as vector representations. This design allows the corpus to be pre-encoded during indexing, enabling user queries to be encoded at retrieval time and relevant passages to be searched from the pre-encoded index, as shown in Figure 1.1. For over half a century, first-stage retrieval solutions have been dominated by sparse vector-based models with heuristic lexical representations, such as BM25 [Robertson et al., 1994]. Even in modern multi-stage ranking architectures [Nogueira and Cho, 2019], which leverage pre-trained transformers [Devlin et al., 2019], these models are typically used as re-rankers over initial candidates retrieved based on BM25 sparse vector representations.

In recent years, advances in pre-trained transformers [Devlin et al., 2019, Liu et al., 2019, Raffel et al., 2020] and approximate nearest neighbor (ANN) search techniques [Guo et al., 2020, Johnson et al., 2021] have established dense retrieval models [Karpukhin et al., 2020, Xiong et al., 2021] as a viable alternative to BM25 for first-stage retrieval. The shift from heuristic-based BM25 to transformer-based dense retrieval models offers two main advantages: (1) Each text is encoded into a low-dimensional semantic vector, enabling query and passage matching even in the absence of direct term overlap. For example, the two texts "Photosynthesis in plants" and "Trees utilize sunlight to convert carbon dioxide and water into glucose and oxygen" lack term overlap but are semantically related. (2) Unlike BM25, dense retrieval models learn to encode queries and passages directly from training data, eliminating the need for manually defined heuristics.

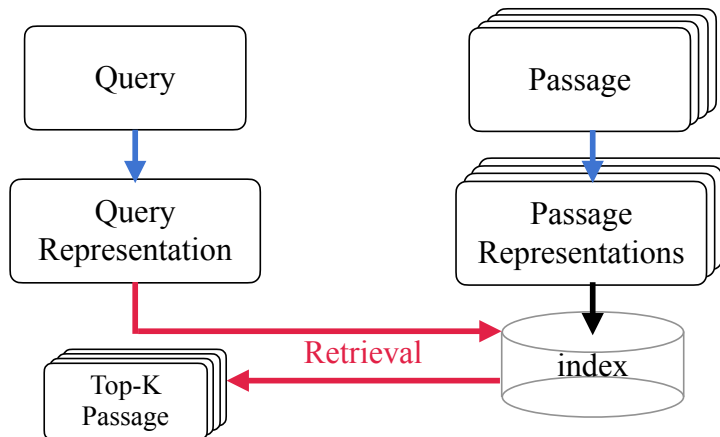


Figure 1.1: An illustration of a first-stage retrieval model.

Robustness across diverse retrieval scenarios is crucial for any retrieval model. However, the advantage of dense retrieval models—being purely data-driven without handcrafted features—also introduces challenges in terms of robustness. Previous work has highlighted the limitations of transformer-based dense retrieval models, especially when compared to BM25 across various tasks, domains, and languages. For instance, Thakur et al. [2021b] demonstrate that state-of-the-art dense retrieval models still lag behind BM25 in domains not seen during training. Similarly, Sciavolino et al. [2021] show that dense retrieval models struggle with simple entity-centric questions, such as “Where was [Entity] born?”. Furthermore, Zhang et al. [2023c] reveal that even when fine-tuned on multilingual training data with high-quality human annotations, dense retrieval models barely outperform BM25 in multilingual search scenarios.

Therefore, deploying a robust dense retrieval model for first-stage retrieval presents significant challenges. In this thesis, we present our efforts to address these challenges. We begin by discussing the two main challenges of deploying a robust dense retrieval model for first-stage retrieval in Section 1.1. Then, in Section 1.2, we present our solutions to these challenges, which will be discussed in detail throughout the thesis.

1.1 Challenges of Deploying a Robust Dense Retrieval Model

1.1.1 Cost at Training

Huge Computation Resources. Training a dense retrieval model is computationally intensive, as contrastive learning with Noise Contrastive Estimation (NCE) loss [Gutmann and Hyvärinen, 2010] requires large batch sizes, which typically demand multiple GPUs. For instance, fine-tuning a standard dense retrieval model [Xiong et al., 2021, Karpukhin et al., 2020] typically requires at least 8 V100 GPUs (32 GB each), and achieving state-of-the-art performance may necessitate 32 or more A100 GPUs (80 GB each) [Qu et al., 2021].

High-Quality Training Data Creation. Since dense retrieval models learn to encode text based on their training data, creating diverse and high-quality training data is crucial for improving their robustness across various retrieval tasks and domains. However, generating such training data solely through human effort is nearly impossible. Even high-quality datasets with human annotations, like MS MARCO [Bajaj et al., 2016], still overlook many relevant passages [Qu et al., 2021]. The standard approach [Ni et al., 2022, Wang et al., 2024] typically involves a two-step training process: first, pre-training a transformer on a large-scale, diverse yet noisy dataset consisting of billions of pairs crawled from the web; and then, fine-tuning it on smaller, high-quality datasets such as MS MARCO and Natural Questions [Kwiatkowski et al., 2019].

1.1.2 Cost at Runtime

To enhance the effectiveness of retrieval systems, two primary hybrid retrieval designs to combine semantic and lexical textual representations are commonly employed: multi-vector bi-encoder models and dense-sparse fusion. Multi-vector bi-encoder models [Khattab and Zaharia, 2020, Gao et al., 2021a, Luan et al., 2021, Li et al., 2023] can be considered bi-encoder hybrid retrieval models since they transform exact term matching into more effective contextualized word embedding matching. Another effective approach is dense-sparse fusion retrieval, where outputs from diverse retrieval systems, such as sparse and dense retrieval models, are combined [Gao et al., 2021b, Lin et al., 2021c]. However, these designs often trade off runtime efficiency in favor of improved effectiveness and robustness.

High Index Storage. Multi-vector bi-encoder models store contextualized token embeddings in the index, requiring around five times the storage of standard dense retrieval models to in-

dex the full MS MARCO passage corpus, as noted by Khattab and Zaharia [2020]. Although advanced compression techniques [Santhanam et al., 2022b] can be applied to reduce the size of the multi-vector index, it still consumes twice the storage of dense retrieval model indexes, which themselves can be further compressed using more advanced techniques [Zhan et al., 2021a, 2022].

High Retrieval Latency. Although dense–sparse fusion is straightforward to implement, maintaining synchronization while searching from the two separate dense and sparse indexes can be challenging. Additionally, hybrid retrieval systems may become significantly less effective when more advanced sparse retrieval models are used [Mackenzie et al., 2023]. For multi-vector bi-encoder models, each query often requires multiple retrievals from the dense index, especially for longer queries, which can further increase the retrieval latency.

1.2 Our Proposed Solutions for Challenges

This thesis addresses the two main challenges outlined in Section 1.1, with the goal of deploying first-stage retrieval models that are robust across various domains, tasks, and languages. Below, we present our proposed solutions to these challenges:

- **Solutions for Saving Computation Resources at Training:** In Chapter 3, we propose an approach to knowledge distillation with tightly-coupled teachers, which improves both training efficiency and effectiveness for dense retrieval models. Incorporated with hard negative mining, our dense retrieval model outperforms existing state-of-the-art dense retrieval models on MS MARCO dataset, which rely on more computationally expensive training strategies. Inspired by the lexical representation pruning method in Chapter 5, in Chapter 6, we introduce an even simple pruning approach and propose Aggretriever, which fully exploits semantic and lexical features extracted from a pre-trained transformer for robust dense retrieval. We demonstrate that with limited training cost on single V100 GPU, Aggretriever is robust to various tasks and domains. Chapter 7 further extends the study of Aggretriever to multilingual search, showing that our proposed mAggretriever, fine-tuned only on English training data, significantly outperforms existing state-of-the-art multilingual dense retrieval models.
- **Solutions for High-Quality Training Data Creation:** In Chapter 4, we introduce a simple strategy to augment diverse high-quality training data. Instead of crawling diverse training data from the web, we propose a technique to automatically augment existing data

from MS MARCO corpus. Our results demonstrate that this approach enables our dense retrieval models to achieve state-of-the-art effectiveness across diverse tasks and domains.

- **Solutions for High Index Storage:** In Chapter 5, we propose a unified dense representation framework for semantic and lexical matching. This framework enables semantic and lexical textual features to be represented in the same low-dimensional dense representations. The framework enables the new bi-encoder model design, $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}$, yielding competitive retrieval effectiveness and significantly reducing index storage.
- **Solutions for High Retrieval Latency:** Similarly, our framework, described in Chapter 5, facilitates efficient dense-sparse fusion retrieval using a single index. This approach supports both approximate nearest neighbor and GPU-based search, reducing retrieval latency.

1.3 Contributions and Outline

In summary, this thesis proposes efficient training techniques and dense-sparse fusion retrieval framework to deploy a dense retrieval model robust to various domains, tasks and languages. The key contributions outlined by the chapters are listed as follows:

1. Chapter 2 introduces the basic concepts of building a retrieval system. We first formalize the main task in the thesis, ad-hoc text retrieval; then, we introduce the two main architectures of ranking models: bi-encoder and cross-encoder rankers. We delve into the different bi-encoder rankers, which are used for first-stage retrieval, including Bag-of-Word models, dense, sparse, dense-sparse fusion and multi-vector retrievers. Subsequently, we introduce the three main procedures to deploy a retrieval model: training, indexing and retrieval. We describe the main datasets used in the thesis for supervised and zero-shot evaluations. Finally, we discuss and compare the existing work related to our thesis.
2. Chapter 3 proposes a novel teacher-student knowledge distillation approach using tightly coupled bi-encoders, namely TCT-ColBERT. This approach enables in-batch knowledge distillation strategy. We demonstrate our proposed knowledge distillation strategy improves both training efficiency and effectiveness (over 10%) for dense retrieval models compared to the standard knowledge distillation approaches which leverage cross-encoder rankers as teachers.
3. Chapter 4 proposes a new data augmentation technique. With the technique, we can easily augment the training data from 0.5 to 100 millions using MS MARCO corpus without crawling data from the web. As a result, our dense retrieval model, DRAGON, achieves

state-of-the-art effectiveness in both supervised and zero-shot evaluations and even outperforms the larger dense retrieval models (with model parameters over 40 times) fine-tuned on billion-scale training pairs crawled from the web.

4. Chapter 5 proposes a novel framework to unify semantic and lexical matching for retrieval, improving the efficiency of deploying fusion retrieval systems. We demonstrate that our framework can be used to conduct efficient hybrid retrieval using any off-the-shelf dense and sparse retrieval models. Additionally, with the proposed framework, we develop a new bi-encoder retriever with semantic and lexical hybrid representations, which is more efficient for deploying than existing transformer-based multi-vector retrieval models.
5. Chapter 6 proposes a new dense retrieval model design, which fully exploits semantic and lexical textual features extracted from a pre-trained transformer to build a robust dense retrieval model. We demonstrate that our model, Aggretriever, significantly outperforms the dense retrieval models fine-tuned in the same condition in both supervised and zero-shot evaluations. The success of Aggretriever not only provides insights on why fine-tuning dense retrieval models is computationally expensive even when initialized from pre-trained transformers, but also offer a solution to deploy a robust dense retrieval model with limited computation resources.
6. Chapter 7 proposes mAggretriever to conquer the multilingual retrieval tasks. We are the first to demonstrate that a dense retrieval model, only fine-tuned on English training data, is able to outperform existing state-of-the-art multilingual bi-encoder retrieval models, which are trained on billion-scale multilingual data crawled from the web.
7. Chapter 8 concludes the thesis and outlines the promising future of building advanced dense retrieval models, leveraging large language models (LLMs) and multimodal LLMs, including incorporating them into retrieval systems for search planning and multimodal retrieval.

Chapter 2

Background

2.1 Ad-Hoc Text Retrieval and Ranking Models

Following the logical framework of [Lin et al., 2021b], we formulate the task of ad-hoc text retrieval as follows: Given a query q , the goal is to retrieve a ranked list of passages $\{p_1, p_2, \dots, p_k\} \in C$ to maximize some ranking metrics, such as nDCG, where C is a collection of passages.¹ Given a relevant query–passage (q and p) pair, the similarity between q and p is denoted as $\text{sim}(q, p)$. In general, ranking models can be categorized into two classes based on how similarity scores are computed.

Bi-Encoder (BE) rankers. As illustrated in Figure 1.1, each query and passage is independently encoded as vector representations, which are then used to compute their similarity score. Formally, given a query–passage (q and p) pair, their similarity score is computed as follows:

$$\text{sim}_{\text{BE}}(q, p) \triangleq \phi_{\text{BE}}(\eta_q^\theta(q), \eta_p^\theta(p)), \quad (2.1)$$

where $\eta_q^\theta(\cdot)$ and $\eta_p^\theta(\cdot)$ denote the encoding functions, parameterized by θ , mapping respective query and passage texts into vector representations, and ϕ_{BE} denotes the scoring function. Note that the parameters of the two encoders can also be independent (i.e., $\eta_p^{\theta_1}, \eta_q^{\theta_2}$) as a Asymmetric bi-encoder. For simplicity, unless otherwise specified, we use Siamese bi-encoder [Bromley et al., 1993] with shared parameters throughout the thesis. We refer readers to Dong et al. [2022] for the discussion of pros and cons of Asymmetric and Siamese bi-encoders.

¹In the thesis, we use “passage” as the unit of document.

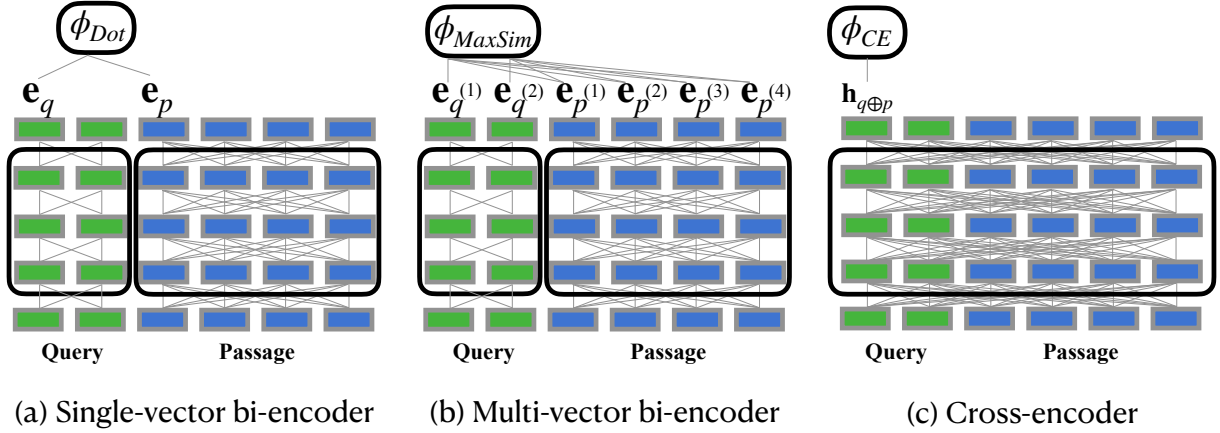


Figure 2.1: Three main architectures of ranking models.

The main advantage of bi-encoder designs is that the vector representations for the passages in the corpus can be pre-computed (prior to retrieval). When retrieving passages in response to a given query, the computationally expensive passage encoding can be skipped, and a fast retrieval algorithm can be used [Mallia et al., 2017, Liu et al., 2004, Malkov and Yashunin, 2020b].

Cross-Encoder (CE) rankers. Compared to bi-encoder rankers, cross-encoder design enables richer interaction between each query and passage by computing their relevance inside a deep neural network [Xiong et al., 2017, Dai et al., 2018, Nogueira and Cho, 2019]. As an illustration, we use the design of leveraging BERT as cross-encoder rankers [Nogueira and Cho, 2019].

$$\text{sim}_{\text{CE}}(q, p) \triangleq \phi_{\text{CE}}^{\theta}(q, p) = W \text{BERT}_{[\text{CLS}]}([\text{CLS}] \oplus p \oplus [\text{SEP}] \oplus q) = W \mathbf{h}_{q \oplus p}, \quad (2.2)$$

where the ranking score is first computed by the hidden representation of the concatenation q and p from BERT (along with the standard special tokens $[\text{CLS}] \oplus p \oplus [\text{SEP}] \oplus q$) and then mapped to a scalar by a mapping matrix W . Note that, different from bi-encoder architecture, the scoring function of cross encoders is parameterized by the trainable θ . The cross-encoder architecture is illustrated in Figure 2.1(c). Despite of its rich interaction between query and passage, the cross-encoder design functions as a re-ranker rather than a retriever, since it is not feasible to compute the interaction between each query and all passages in the corpus. Thus, a standard multi-stage pipeline [Nogueira and Cho, 2019] is to first use a bi-encoder ranker to retrieve top- k passages as candidates and leverage the cross-encoder ranker to output the final ranking among the candidates.

2.2 Bi-Encoder Retrieval Models

In the thesis, we will focus on building a robust retrieval model; thus, we further discuss two main retrieval models based on bi-encoder architecture: single- and multi-vector retrievers.

2.2.1 Single-Vector Bi-Encoder Retrievers

As illustrated in Figure 2.1(a), each input (query or passage) is encoded into a single-vector representation; then, dot product is applied to the query and passage vectors to compute their relevance score:

$$\phi_{\text{Dot}}(\eta_q^\theta(q), \eta_p^\theta(p)) = \langle \eta_q^\theta(q), \eta_p^\theta(p) \rangle = \langle \mathbf{q}, \mathbf{p} \rangle, \quad (2.3)$$

where $\mathbf{q}(\mathbf{p}) \in \mathbb{R}^d$ are the encoded query (passage) single-vector representations, respectively.

Bag-of-Words Models. Traditional Bag-of-Words (BoW) retrieval models dominate over half a century, which estimate the relevance between queries and passages based on the overlap of important terms. For example, TF-IDF first estimates the importance (or weight) of each term appearing in a passage through its normalized term frequency multiplied by inverse document frequency. Given a query, its relevance score to the passage is to sum over the weights of terms, which co-occur in the query and passage.

The more effective BoW model, BM25, can be considered an improved version of TF-IDF. The relevance score between a query and passage is computed as follows:

$$\sum_{q_i \in q} \text{IDF}(q_i) \cdot \frac{f(q_i, p) \cdot (k_1 + 1)}{f(q_i, p) + k_1 \cdot (1 - b + b \cdot \frac{|p|}{\text{avg}(|p|)})}, \quad (2.4)$$

where $f(q_i, p)$ is the term frequency of q_i term in p , $|p|$ is the length (i.e., # of terms) of passage p , $\text{avg}(|p|)$ is the averaged length of all the passages in the given corpus, and k_1 and b are hyperparameters. Alternatively, we can consider BM25 an heuristic encoder, where each query and passage are treated bags of words and respectively encoded into high-dimensional lexical vectors: $\mathbf{q}_{\text{BoW}}$ and $\mathbf{p}_{\text{BoW}}$. The query is encoded into $\mathbf{q}_{\text{BoW}} = \sum_{q_i \in q} \hat{\mathbf{u}}(q_i) \in \mathbb{R}^{|V_{\text{BoW}}|}$, where $\hat{\mathbf{u}}(q_i)$ is the unit vector with non-zero entry at the dimension corresponding to the term q_i , and V_{BoW} is the set of all the appearing terms in the corpus. The passage is encoded into $\mathbf{p}_{\text{BoW}} = \sum_{p_i \in p} \hat{\mathbf{u}}(p_i) \cdot \text{IDF}(p_i) \cdot \frac{f(p_i, p) \cdot (k_1 + 1)}{f(p_i, p) + k_1 \cdot (1 - b + b \cdot \frac{|p|}{\text{avg}(|p|)})}$. Note that $\mathbf{q}_{\text{BoW}}$ and $\mathbf{p}_{\text{BoW}}$ are usually sparse vectors with millions of dimensions, where only tens to hundreds of entries are non-zero.

Since pre-trained transformers [Devlin et al., 2019, Liu et al., 2019, Raffel et al., 2020] have demonstrated superior capabilities in understanding human natural language, the focus of research has shifted from human-defined features (such as TF-IDF and BM25) to fine-tuning transformer-based single-vector bi-encoder models. Two main categories of learned bi-encoder retrievers have been proposed: learned dense and sparse retrieval models [Lin, 2022]. It is worth noting that there were many relevant works [Huang et al., 2013, Zamani et al., 2018] prior to the advent of transformers [Vaswani et al., 2017]. In this thesis, for convenience, we refer to both dense and sparse retrieval models as learned transformer-based retrievers.

Dense Retrieval Models. Reimers and Gurevych [2019] first explore to fine-tune a pre-trained transformer, such as BERT, to encode sentences into single-vector representations and perform dot product over the vectors to measure sentence similarity. The subsequent work [Zhan et al., 2020b, Karpukhin et al., 2020] adopts the same strategy to text retrieval, where the measurement of sentence similarity is replaced by query–passage similarity. Normally, the dense retrieval models encode each text into a 768-dimensional semantic vector from the $[\text{CLS}]$ vector (i.e., $\mathbf{q}_{[\text{CLS}]}, \mathbf{p}_{[\text{CLS}]} \in \mathbb{R}^{768}$) or the pooled one averaged over all the contextualized token embeddings.

Sparse Retrieval Models. One early technique, DeepCT [Dai and Callan, 2020], uses a linear layer to project BERT token embeddings into contextualized term weights for input tokens. Subsequent work [Mallia et al., 2021, Lin and Ma, 2021, Zhuang and Zuccon, 2021b] further combines DeepCT and other document expansion methods [Nogueira and Lin, 2019, Zhuang and Zuccon, 2021a] to address vocabulary mismatch and missing terms in DeepCT. For example, DeepImpact [Mallia et al., 2021] uses a trained sequence-to-sequence model [Nogueira and Lin, 2019] to expand the original passages in the corpus and learn contextualized term weights. Other models [Lin and Ma, 2021, Zhuang and Zuccon, 2021b] follow a similar approach to DeepImpact but term matching is performed in the space of BERT wordpiece tokens rather than whole words. They generate term weights only for tokens appearing in each (possibly expanded) query or passage, and thus the lexical representations are sparse by design. However, to achieve competitive effectiveness, these techniques require additional models for term expansion (see Zhuang and Zuccon [2021b] for more discussion). In contrast, some researchers [Formal et al., 2021b, Bai et al., 2020] use the masked language model (MLM) layer in transformers such as BERT to learn term weighting and expansions at the same time. For these approaches, sparsity regularization must be applied during model training to ensure that the generated lexical representations are amenable to retrieval using inverted indexes.

Generally, these neural approaches to lexical term matching can be viewed as projecting queries and passages into high-dimensional lexical vectors, $\mathbf{q}_{\text{BoW}}$ and $\mathbf{p}_{\text{BoW}} \in \mathbb{R}^{|V_{\text{wp}}|}$, where $|V_{\text{wp}}|$ is

the vocabulary size of wordpiece tokens. The value in each dimension refers to each token’s term weight, similar to the aforementioned BoW models. As with dense retrieval, the relevance score between a query and a passage is computed by the inner product of their vector representations.

2.2.2 Multi-Vector Bi-Encoder Retrievers

To enhance the interaction between text representations, some studies propose encoding each text into multi-vector representations and applying more sophisticated scoring functions. As shown in Figure 2.1(b), one of the most representative approaches is ColBERT [Khattab and Zaharia, 2020]. In ColBERT, each query or passage is encoded into a sequence of contextualized word-piece token embeddings, and the MaxSim operation is applied to capture fine-grained relevance between the query and the passage:

$$\phi_{\text{MaxSim}}(\eta_q^\theta(q), \eta_p^\theta(p)) = \sum_{\mathbf{q}_i \in \eta_q^\theta(q)} \max_{\mathbf{p}_j \in \eta_p^\theta(p)} \langle \mathbf{q}_i, \mathbf{p}_j \rangle, \quad (2.5)$$

where $\mathbf{q}_i$ and $\mathbf{p}_j$ are vector representations of the i - and j -th query and passage tokens, respectively. The subsequent work [Gao et al., 2021a, Hofstätter et al., 2022, Li et al., 2023] further improves retrieval systems based on multi-vector bi-encoder architecture. These designs can be considered bi-encoder hybrid retrieval models since it combines the advantages of dense (semantic) and sparse (lexical) retrieval models.

2.3 Training Bi-Encoder Retrieval Models

Contrastive learning is a standard text representation learning paradigm for bi-encoder rankers. Specifically, given a query q , corpus C and its relevant passage $p^+ \in C$, a model is trained by minimizing negative log likelihood (NLL) of the positive $\{q, p^+\}$ pair over all the passages in C :

$$NLL = -\log \frac{\exp(\phi(\eta_q^\theta(q), \eta_p^\theta(p^+)))}{\sum_{p' \in C} \exp(\phi(\eta_q^\theta(q), \eta_p^\theta(p'))) \quad (2.6)$$

$$\approx NCE = -\frac{1}{|\mathcal{B}|} \sum_{n=1}^{|\mathcal{B}|} \log \frac{\exp(\phi(\eta_q^\theta(q_n), \eta_p^\theta(p_i^+)))}{\sum_{p' \in \mathcal{D}_B} \exp(\phi(\eta_q^\theta(q), \eta_p^\theta(p'))) \quad (2.7)$$

The main challenge of evaluating and computing gradients of Eq. (2.6) is the prohibitively expensive computation cost given the number of passages in the corpus C , typically millions

or even more [Chang et al., 2020]. Thus, previous work approximates Eq. (2.6) by Noise-Contrastive Estimation [NCE; Gutmann and Hyvärinen, 2010]. Specifically, given a training triplet (q, p^+, p^-) , C is replaced with a small subset $\mathcal{D} = \{p^+, p^-\}$, where p^- is from a noisy distribution such as candidates retrieved by BM25 [Nogueira and Cho, 2019], retrieved by an asynchronously updated bi-encoder model itself [Xiong et al., 2021] or filtered by fine-tuned transformers [Qu et al., 2020]. These approaches can be further combined with in-batch negative sampling, as used by Karpukhin et al. [2020], where C is replaced by $\mathcal{D}_B = (p_1^+ \cdots p_{|\mathcal{B}|}^+ \cdots p_1^- \cdots p_{|\mathcal{B}|}^-)$ as shown in Eq. (2.7). NCE with in-batch negatives makes an assumption that all the passages in a mini-batch, except for p_n^+ , are negative samples for q_n . This technique effectively increases the number of negative samples for q_n from 1 to $2 \cdot |\mathcal{B}| - 1$.

2.4 Indexing and Retrieval

There are various fast retrieval approaches, each associated with its own indexing method. Although single-vector bi-encoder models use the inner product between two vectors to measure their similarity, dense retrieval and sparse (or BoW) retrieval models employ different indexing and retrieval strategies. These approaches are outlined as follows.

2.4.1 Sparse Retrieval with Inverted Index

The inverted index is the standard structure used to store a corpus of documents (or passages) for subsequent sparse retrieval, and it consists of two main components: a dictionary and posting lists. The dictionary contains all the unique terms that appear in the corpus, while the postings lists record each term’s occurrences in the corpus. There are two steps to create inverted index from a corpus of documents: (1) The content of a set of documents is parsed into terms (or keywords). This typically involves tokenization (breaking the text into individual words) and sometimes normalization (converting words to lowercase, stemming, removing stopwords, etc.). (2) For each word (or token) in a document, the system records the word and the document ID (or other document metadata, such as TF and IDF). This process creates a list of terms, and for each term, a corresponding “postings list” that contains the documents in which the term appears.

While retrieval, given a query, which is first parsed by the same tokenizer as the corpus, the posting lists of each occurring tokens in the query are retained so that the relevance scores between the query and the document candidates in the set of posting lists can be computed. There are various methods to efficiently search top- k candidates from inverted index, such as document-at-a-time (DAAT), score-at-a-time (SAAT) query evaluations and the more advanced ones, which

combine DAAT and SAAT, such as Block-Max WAND. We refer readers to Tonellotto et al. [2018] for more detail.

2.4.2 Dense Retrieval with Approximate Nearest Neighbor Search

Compared to sparse retrieval, dense retrieval aims to compute the inner product between vectors with relatively lower dimensions, typically below 1K, as opposed to over 30K in sparse retrieval. In addition to directly loading the entire corpus index into GPUs for fast retrieval, several approximate nearest neighbor (ANN) search techniques exist, which sacrifice retrieval accuracy in favor of lower query latency. We list three common techniques, all of which are well supported by existing ANN search libraries [Guo et al., 2020, Johnson et al., 2021].

Product Quantization (PQ). PQ [Jégou et al., 2011] reduces the memory footprint and accelerates the search process in high-dimensional spaces. It works by decomposing the original high-dimensional vectors into smaller, low-dimensional subspaces, and then quantizing each subspace independently using a codebook. The combination of these quantized sub-vectors forms a compact representation of the original data. During the search, distances are computed using these compressed representations, allowing for fast and efficient retrieval with minimal loss in accuracy. PQ is widely used in large-scale search applications due to its balance between speed and accuracy.

Hierarchical Navigable Small World (HNSW). HNSW [Malkov and Yashunin, 2020a] is a graph-based algorithm used for time-efficient ANN search. It builds a multi-layer graph where each node represents a data point, and edges connect nodes based on proximity. The structure is hierarchical, with higher layers containing fewer nodes and longer connections for global navigation, while lower layers provide more detailed, local connections. During the search, the algorithm starts from the top layer and navigates through the graph by moving toward the nearest neighbor in each layer until it reaches the most similar points.

Inverted File (IVF). IVF is adapted as a clustering-based technique for efficient ANN search, commonly paired with PQ. The core idea is to partition the dense vector space into several clusters during an offline phase, and each cluster corresponds to an entry in the inverted file. When a query is processed, it is first mapped to the nearest cluster(s), reducing the search space to only a subset of the vectors associated with those clusters. This method is particularly effective for large-scale dense vector retrieval, as it narrows down the search scope before applying more precise retrieval techniques, such as Product Quantization, within each cluster.

2.4.3 Dense–Sparse Hybrid Retrieval

In theory, we can concatenate any lexical (from a BoW or sparse retrieval model) and semantic (from a dense retrieval model) vectors into a single vector and apply inner product to compute the relevance score between any query and passage.

$$\text{sim}_{\text{hybrid}}(q, p) \triangleq \langle (\lambda \cdot \mathbf{p}_{\text{BoW}} \oplus \mathbf{p}_{[\text{CLS}]}) , (\lambda \cdot \mathbf{q}_{\text{BoW}} \oplus \mathbf{p}_{[\text{CLS}]}) \rangle, \quad (2.8)$$

where λ is a hyperparameter to control the weighting between lexical and semantic matching when computing relevance scores. However, in practical, text retrieval with lexical representations is usually performed using inverted indexes due to their high dimensionality, while text retrieval using semantic representations is usually performed under completely different infrastructure. Alternatively, for each query q , we use its lexical and semantic representations to retrieve top- k passages, IV_{topk} and ANN_{topk} , with inverted index and ANN search, respectively. Then, we compute the score for each retrieved passage ($p \in IV_{\text{topk}} \cup ANN_{\text{topk}}$) as follows:

$$\begin{cases} \lambda^2 \cdot \langle \mathbf{q}_{\text{BoW}}, \mathbf{p}_{\text{BoW}} \rangle + \min_{p' \in ANN_{\text{topk}}} \langle \mathbf{q}_{[\text{CLS}]}, \mathbf{p}'_{[\text{CLS}]} \rangle, & \text{if } p \notin ANN_{\text{topk}} \\ \min_{p' \in IV_{\text{topk}}} \lambda^2 \cdot \langle \mathbf{q}_{\text{BoW}}, \mathbf{p}'_{\text{BoW}} \rangle + \langle \mathbf{q}_{[\text{CLS}]}, \mathbf{p}_{[\text{CLS}]} \rangle, & \text{if } p \notin IV_{\text{topk}} \\ \lambda^2 \cdot \langle \mathbf{q}_{\text{BoW}}, \mathbf{p}_{\text{BoW}} \rangle + \langle \mathbf{q}_{[\text{CLS}]}, \mathbf{p}_{[\text{CLS}]} \rangle, & \text{otherwise,} \end{cases} \quad (2.9)$$

Eq. (2.9) manages to compute the approximated relevance scores between the query and those passages, which are not retrieved by either inverted index or ANN search. The approximation of the hybrid retrieval is proposed in our previous work [Lin et al., 2021c].

2.5 Datasets and Evaluation Metrics for Text Retrieval

In most part of our thesis, following previous work [Izacard et al., 2022, Xiao et al., 2022, Yu et al., 2022, Formal et al., 2022, Santhanam et al., 2022b], we consider MS MARCO [Bajaj et al., 2016] as supervised data and BEIR benchmark [Thakur et al., 2021b] for zero-shot evaluations. We detail the datasets and corresponding evaluation metrics as follows.

Datasets for Supervised Evaluations. MS MARCO passage ranking dataset [Bajaj et al., 2016] comprises a corpus of 8.8M web passages and a set of ~ 0.5 M relevant (query, passage) pairs as training data. There are three public query sets for evaluation detailed as follows: (a) MS MARCO Dev: 6980 queries comprise the development set for the MS MARCO passage

leaderboard, with on average one relevant passage per query. Following established procedure [Nogueira and Cho, 2019], we report MRR@10 and R@1000 as the evaluation metrics. (b) TREC DL [Craswell et al., 2019, 2020]: the organizers of the 2019 (2020) Deep Learning Track at the Text REtrieval Conference (TREC) released 43 (53) queries with graded relevance labels, where (query, passage) pairs were annotated by NIST assessors. We report nDCG@10 for these two evaluation sets.

Datasets for Zero-Shot Evaluations. We use BEIR benchmark, introduced by Thakur et al. [2021b], which contains 18 distinct IR datasets spanning diverse domains and tasks, including retrieval, question answering, fact checking, question paraphrasing, and citation prediction. Each individual dataset comprises its own corpus, queries, and relevance judgments. The BEIR benchmark consists of 14 public datasets (namely, TREC-COVID [Voorhees et al., 2021], NF-Corpus [Boteva et al., 2016], FiQA-2018 [Maia et al., 2018], ArguAna [Wachsmuth et al., 2018a], Tóuche-2020, Quora, SCIDOCS [Cohan et al., 2020], SciFact [Wadden et al., 2020], NQ [Kwiatkowski et al., 2019], HotpotQA [Yang et al., 2018], DBPedia [Hasibi et al., 2017], FEVER [Wachsmuth et al., 2018b], Climate-FEVER [Diggelmann et al., 2020] and CQADupStack [Hoogeveen et al., 2015]) and 4 private datasets (namely, Robust04 [Voorhees, 2005], Signal-1M [Suarez et al., 2018], TREC-NEWS [Soboroff et al., 2020] and BioASQ [Nentidis et al., 2023]). In order to compare with previous work [Formal et al., 2021a, Santhanam et al., 2022b], in some experiments, we only conduct evaluation on the full public datasets or 13 out of the 14 public datasets (excluding CQADupStack), denoted BEIR-14 and BEIR-13, respectively. Otherwise, BEIR denotes the full 18 datasets. Model retrieval effectiveness is evaluated in terms of nDCG@10.

2.6 Related Work

In this thesis, we focus on improving the robustness of dense retrieval in two main directions: better training strategies and semantic–lexical hybrid representation learning. We discuss and compare the existing work related to the two main directions in the following.

2.6.1 More Sophisticated Training Techniques for Dense Retrieval

Knowledge Distillation. Our thesis is closely related to the previous work exploring knowledge distillation [Hinton et al., 2015] using cross encoder to improve the effectiveness of dense retrieval [Hofstätter et al., 2020, Qu et al., 2021]. However, they only take the advantage of soft

labels from knowledge distillation and use the relevant passages labeled by humans. The more recent work [Ren et al., 2021, Zeng et al., 2022] mines more positive samples using cross encoder to further augment the limited relevance labels by humans. Nevertheless, it is challenging for cross encoders to augment relevance labels for queries in scale due to its low efficiency. Chen et al. [2022] first explore label augmentation using single sparse retrieval model on large-scale queries and demonstrate that a dense retriever can mimic a teacher of a sparse retriever (e.g., BM25).

Dense Retrieval Pre-Training. There are two popular approaches to pre-training a dense retriever. The first one is contrastive pre-training, aiming to increase the size of training data by creating artificial text pairs [Lee et al., 2019, Chang et al., 2020, Izacard et al., 2022] from a corpus or collecting question-answer pairs [Oguz et al., 2022, Ni et al., 2022] from websites. The second one is masked auto encoding pre-training, where models are trained to recover the corrupted texts [Gao and Callan, 2021, Lu et al., 2021, Xiao et al., 2022, Wu et al., 2023].

Different from the prior work, in Chapter 3, we propose to use the bi-encoder model, ColBERT, as a more efficient teacher compared to cross-encoder. Additionally, in Chapter 4, we further propose using multiple bi-encoder models as teachers to augment the training data with an easy-to-hard manner. More importantly, our data augmentation approach does not require additional efforts to crawl data from the web.

2.6.2 Bi-Encoder Hybrid Retrieval Models

Our work in Chapters 5, 6, and 7 can be characterized as hybrid representation learning. Recent work [Gao et al., 2021a, Hofstätter et al., 2022, Li et al., 2023, Shen et al., 2023] proposes jointly training [CLS] and token-level representations to fuse semantic and lexical matching. These prior works require customized implementations for top- k retrieval. Similar to the prior work, in Chapter 5, we propose and implement a dense representation framework to facilitate fusion retrieval with improved effectiveness and efficiency. In Chapters 6 and 7, we further propose fusing semantic and lexical textual representations into a dense vector for retrieval. This enables retrieval in a simple execution environment (e.g., Faiss or ScaNN) without the need for additional customized implementation. Additionally, we are the first to extend hybrid representation learning to the multilingual retrieval scenario.

Chapter 3

Efficiently Training a Dense Retrieval Model

3.1 Overview

Transformer-based bi-encoder dense retrieval models are widely used for learning text representations in text retrieval tasks. However, prior work has shown that fine-tuning such dense retrieval models requires substantial computational resources (e.g., at least 8 V100 GPUs with 32 GB memory each), making them impractical for many academic researchers. In this chapter, we present an efficient approach to training effective dense retrieval models by leveraging knowledge distillation from the ColBERT late-interaction ranking model. Specifically, we propose transferring knowledge from a strong bi-encoder teacher model to a student model by distilling ColBERT’s expressive MaxSim operator into a simple dot product. This bi-encoder teacher–student setup offers the advantage of efficiently incorporating in-batch negatives during knowledge distillation, enabling richer interactions between the teacher and student models. Moreover, using ColBERT as the teacher reduces training costs without compromising effectiveness, compared to using a cross-encoder teacher. Experiments on the MS MARCO passage and document ranking tasks demonstrate that our approach enables models to learn robust representations for dense retrieval efficiently. Our training method is feasible using a single TPU-v2 (64 GB) on free Google Colab or two V100 GPUs (32 GB each). These results are detailed in our work on TCT-ColBERT [Lin et al., 2021c].

3.2 Methodology: TCT-ColBERT

3.2.1 Knowledge Distillation

Knowledge distillation [Hinton et al., 2015] has been shown to be effective across various fields. In particular, Barkan et al. [2020] demonstrate the success of distilling a trained cross-encoder into a bi-encoder for sentence similarity tasks. In the field of information retrieval, Hofstätter et al. [2020] show that using a cross-encoder teacher for knowledge distillation significantly enhances the effectiveness of single-vector bi-encoders for dense retrieval. Generally, prior approaches to knowledge distillation aim to train the single-vector bi-encoder model to mimic the probability distribution of the more powerful cross-encoder teacher model, as follows:

$$P^\theta(p, q, C) = \frac{\exp(\phi_{\text{Dot}}(\eta_q^\theta(q), \eta_p^\theta(p)))}{\sum_{p' \in C} \exp(\phi_{\text{Dot}}(\eta_q^\theta(q), \eta_{p'}^\theta(p')))} \approx \frac{\exp(\phi_{\text{CE}}^{\hat{\theta}}(q, p)/\tau)}{\sum_{p' \in C} \exp(\phi_{\text{CE}}^{\hat{\theta}}(q, p')/\tau)} = P^{\hat{\theta}}(q, p, C), \quad (3.1)$$

where $\phi_{\text{Dot}}(\eta_q^\theta(q), \eta_p^\theta(p))$ and $\phi_{\text{CE}}^{\hat{\theta}}(q, p)$ denotes the relevance score between q and p estimated by the single-vector bi-encoder student and the more powerful cross-encoder teacher model as shown in Eq (2.1) and (2.2), respectively; τ is the temperature hyperparameter used in the knowledge distillation framework [Hinton et al., 2015]. However, we argue that using cross-encoder as a teacher model is computationally expensive; thus, making the knowledge distillation procedure less sample efficient and less practical.

This chapter introduces a simple yet effective technique for efficiently incorporating in-batch negative samples during knowledge distillation when training a single-vector bi-encoder model. For convenience, we refer to this technique as “in-batch knowledge distillation” throughout the chapter. We empirically demonstrate that our model, even when trained with BM25 negatives, can outperform models trained using a more powerful cross-encoder as a teacher. When hard negatives are used, our method approaches state-of-the-art performance in dense retrieval. Additionally, our in-batch knowledge distillation technique allows for the inclusion of hard negatives in a computationally efficient manner, without the need for large GPU memory to support large batch sizes or costly periodic index refreshes.

3.2.2 Knowledge Distillation with In-Batch Samples

Using knowledge distillation in Eq. (3.1) provides soft labels for single-vector bi-encoder training, and can be integrated with the previously mentioned NCE framework in Eq. (2.7). Specif-

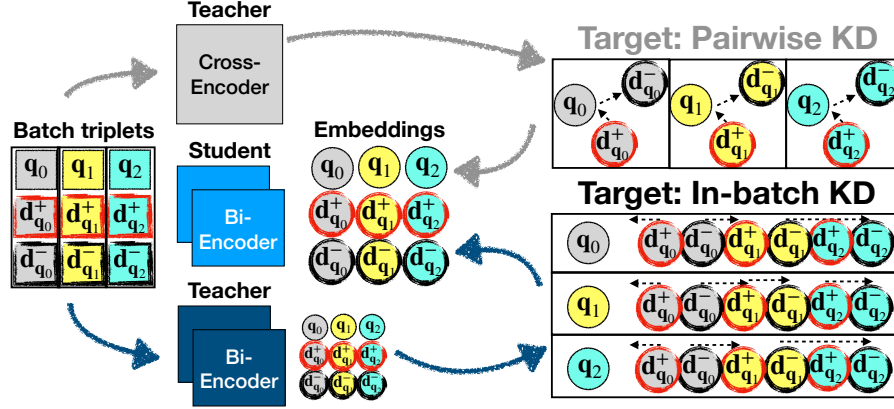


Figure 3.1: Illustration of the differences between pairwise knowledge distillation (KD) and our proposed in-batch knowledge distillation.

ically, given a mini-batch $\mathcal{B}$ of triplets $(q_n, p_n^+, p_n^-)_{\{n|(1 \leq n \leq |\mathcal{B}|)\}}$, we estimate $P^\theta(p, q, C)$ on in-batch examples guided by an auxiliary teacher model $P^{\hat{\theta}}(p, q, C)$ through the minimization of Kullback–Leibler (KL) divergence of the two distributions:

$$\mathcal{L}_{\text{in-batch}} = \arg \min_{\theta} \frac{1}{|\mathcal{B}|} \sum_{n=1}^{|\mathcal{B}|} \sum_{p' \in \mathcal{D}_{\mathcal{B}}} P^{\hat{\theta}}(q_n, p', \mathcal{D}_{\mathcal{B}}) \log \frac{P^{\hat{\theta}}(q_n, p', \mathcal{D}_{\mathcal{B}})}{P^\theta(q_n, p', \mathcal{D}_{\mathcal{B}})}. \quad (3.2)$$

Note that here we consider all pairwise relationships between queries and passages within a mini-batch that contains a query set $\{q_1 \cdots q_{|\mathcal{B}|}\}$ and a passage set $\mathcal{D}_{\mathcal{B}} = \{p_1^+ \cdots p_{|\mathcal{B}|}^+ \cdots p_1^- \cdots p_{|\mathcal{B}|}^-\}$. Also note that we replace C in Eq. (3.1) with $\mathcal{D}_{\mathcal{B}}$ as an approximation to make the computation feasible. This approach is discussed in Section 2.3 and has been shown to be effective in prior studies [Chang et al., 2020, Karpukhin et al., 2020].

3.2.3 Choices of Teacher Models

Cross-Encoder. Prior studies [Hofstätter et al., 2020, Gao et al., 2020] have demonstrated that a well-fine-tuned cross-encoder serves as an effective teacher, as it enables rich interactions between the intermediate transformer representations of a query q and a passage p . For instance, a ”vanilla” cross-encoder design using BERT can be expressed as Eq. (2.2), where the ranking score is computed from the hidden representation of the concatenation $q \oplus p$ (including the standard special tokens) produced by BERT and subsequently mapped to a scalar via a pooling operation with a mapping matrix W .

Table 3.1: Training cost comparison. We report the training time per batch against the baseline (without a teacher model) on a single TPU-v2. Our backbone model is BERT-base, with batch size 96. The in-batch cross-encoder training time is not available because it exceeds the memory limit.

Teacher	knowledge distillation strategy	
	Pairwise ($\mathcal{L}_{\text{pairwise}}$)	In-batch ($\mathcal{L}_{\text{in-batch}}$)
Cross-encoder	+48.1%	OOM
ColBERT	+32.7%	+33.5%

Although effective, due to BERT’s quadratic complexity with respect to input sequence length, this design makes exhaustive combinations between a query and possible candidates impractical, since this requires $|\mathcal{B}|^2$ inference times from the cross-encoder teacher to compute $P^{\hat{\theta}}(q, p, \mathcal{D}_{\mathcal{B}})$ in Eq. (3.2). In addition, the computation of $P^{\hat{\theta}}(q, p, \mathcal{D}_{\mathcal{B}})$ in Eq. (3.1) itself is expensive. Thus, an alternative is to further simplify Eq. (3.2) as follows:

$$\mathcal{L}_{\text{pairwise}} = \arg \min_{\theta} \frac{1}{|\mathcal{B}|} \sum_{n=1}^{|\mathcal{B}|} \sum_{p' \in \mathcal{D}_n} P^{\hat{\theta}}(q_i, p', \mathcal{D}_n) \log \frac{P^{\hat{\theta}}(q_n, p', \mathcal{D}_n)}{P^{\theta}(q_n, p', \mathcal{D}_n)}, \quad (3.3)$$

where $\mathcal{D}_n = \{p_n^+, p_n^-\}$ denotes the pairwise (positive and negative) passage set for q_n . Eq. (3.3) computes the KL divergence of only two probabilities of a positive pair (q_i, p_i^+) and a negative pair (q_i, p_i^-) , which we term *pairwise* knowledge distillation. This simplification is adopted by prior studies [Hofstätter et al., 2020]. However, this might not yield a good approximation of Eq. (3.1).

Bi-Encoder. A bi-encoder can also be leveraged as a teacher model, which has the advantage that it is more feasible to perform exhaustive comparisons between queries and passages since they are passed through the encoder independently. As discussed in Section 2.1, among bi-encoder designs, ColBERT is a representative model that uses fine-grained MaxSim interaction between multi-vector query and passage representations, as shown in Eq. (2.5). This design improves the robustness of dense retrieval compared to the single-vector bi-encoder design. Thus, ColBERT is a suitable teacher for in-batch knowledge distillation compared to the cross-encoder teacher, where only pairwise knowledge distillation can be used.

The main contribution of this work is in-batch knowledge distillation with a tightly coupled teacher. By leveraging a bi-encoder as the teacher, our approach enables enriched interactions between the teacher and student through in-batch knowledge distillation, as defined in

Eq. (3.2). This process requires exhaustive inference over all query–passage combinations in the mini-batch $\mathcal{B}$ but incurs only a computation cost of $2 \cdot |\mathcal{B}|$, compared to $|\mathcal{B}|^2$ when using a cross-encoder teacher. We refer to this design as *Tightly-Coupled Teacher with ColBERT* (TCT-ColBERT). Table 3.1 compares the training costs of different teacher models. Cross-encoders exhibit the highest training cost when training with pairwise knowledge distillation. In contrast, ColBERT enables in-batch knowledge distillation with a significantly reduced training cost relative to pairwise knowledge distillation. Figure 3.1 illustrates the comparison between pairwise and in-batch knowledge distillation, highlighting the differences in computational cost when using cross-encoder and ColBERT teachers, respectively.

TCT-ColBERT offers a flexible design for bi-encoders, provided the encoders independently generate query and passage representations. For simplicity, our student model employs shared encoder weights for both queries and passages, similar to the teacher model, ColBERT. Following Khattab and Zaharia [2020], we prepend a [CLS] token and a special [Q] ([D]) token to the input sequence of each query (passage) for both the teacher and student models. The student encoder generates single-vector dense representations ($\mathbf{e}_q, \mathbf{e}_p$) by applying average pooling to the token embeddings from the final layer.

3.3 Experimental Setups

In this section, we conduct experiments on the MS MARCO passage and document corpora. For passage ranking, we first train models using negatives mined by BM25 as a warm-up and compare different knowledge distillation methods. We then further train the models using hard negatives retrieved from the BM25-warmed checkpoint. For document ranking, following prior work [Xiong et al., 2021, Zhan et al., 2020a, Lu et al., 2021], we start with the BM25-warmed checkpoint from passage ranking and perform additional hard negative training.

3.3.1 Passage Retrieval

We perform passage retrieval on the MS MARCO passage ranking dataset [Bajaj et al., 2016]. We train models on the training queries and evaluate model effectiveness on two test sets of queries: MARCO Dev and TREC-DL ’19. We refer readers to Section 2.5 for more details. To compare with current state-of-the-art models, we evaluate our design, TCT-ColBERT, using two approaches for negative sampling: BM25 and hard negatives retrieved by the bi-encoder itself.

3.3.2 Document Retrieval

To validate the effectiveness and generality of our training strategy, we conduct further experiments on document retrieval using the MS MARCO document ranking dataset. This dataset contains 3.2M web pages gathered from passages in the MS MARCO passage ranking dataset. Similar to the passage condition, we evaluate model effectiveness on two test sets of queries: (1) MARCO Dev: the development set contains 5193 queries, each with exactly one relevant document. (2) TREC-DL '19: graded relevance judgments are available from the TREC 2019 Deep Learning Track, but only for 43 queries. Per official guidelines, we report different metrics for the two query sets: MRR@100 for MARCO Dev and NDCG@10 for TREC-DL '19.

Following the FirstP setting for document retrieval described in [Xiong et al., 2021], we feed the first 512 tokens of each document for encoding, and start with the warmed-up checkpoint for our encoder’s parameters trained for passage retrieval (using BM25 negatives, as described in Section 3.3). The settings for fine-tuning our warmed-up encoder (e.g., learning rate, training steps, top-200 negative sampling) are the same as those for passage retrieval except for batch size, which is set to 64.

3.4 Experimental Results

3.4.1 Passage Retrieval

Training with BM25 Negatives. In this setting, models are trained using the official public data, where negative samples are produced by BM25.¹ We compare different bi-encoder models using BERT-base as the backbone, which uses single 768-dimensional vectors to represent each query and passage:

1. Baseline: a single-vector bi-encoder trained with in-batch negatives, as discussed in Section 2.3, which is similar to Karpukhin et al. [2020] but uses a smaller batch size.
2. Pairwise knowledge distillation: the approach of Hofstätter et al. [2020], who improve ranking effectiveness using cross-encoders with pairwise knowledge distillation.

We also compare against two models, KD-T1 and KD-T2, which use BERT-base bi-encoders as student models. In the former, the student is distilled from a BERT-base cross-encoder, while

¹<https://msmarco.z22.web.core.windows.net/msmarcoranking/triples.train.small.tar.gz>

Table 3.2: Passage retrieval results with BM25 negative training. For knowledge distillation (KD) methods, the effectiveness of teacher (T) models is also reported. All our implemented models are labeled with a number and superscripts represent significant improvements over the labeled model (paired t -test, $p < 0.05$). Note that Bonferroni correction is not applied, so the significance tests may risk overconfidence.

Strategy	Model	Teacher		MARCO Dev		DL19
		Model	# params	MRR@10 (T/S)	R@1K	NDCG@10 (T/S)
-	(1) Baseline	-	-	- / 0.310	0.945	- / 0.626
Pairwise KD	KD-T1 [Hofstätter et al., 2020]	CE	110M	0.376 / 0.304	0.931	0.730 / 0.631
	KD-T2 [Hofstätter et al., 2020]	3×CE	467M	0.399 / 0.315	0.947	0.743 / 0.668
	(2) KD-T2 (Ours)	3×CE	467M	0.399 / 0.341 ¹	0.964 ¹	0.743 / 0.659 ¹
	(3) KD-ColBERT	ColBERT	110M	0.350 / 0.339 ¹	0.962 ¹	0.730 / 0.670 ¹
In-batch KD	(4) TCT-ColBERT	ColBERT	110M	0.350 / 0.344 ^{1,3}	0.967 ^{1,3}	0.730 / 0.685 ¹

the latter is distilled from ensemble cross-encoders comprising BERT-base, BERT-large, and ALBERT-large. The figures reported in Table 3.2 are copied from Hofstätter et al. [2020]. For a fair comparison with our models based on KL-divergence knowledge distillation, we also implement our KD-T2 using the precomputed pairwise softmax probabilities provided by Hofstätter et al. [2020] (who use MSE margin loss for knowledge distillation). In addition, we adopt *pairwise* softmax probabilities from fine-tuned ColBERT to train KD-ColBERT for comparison.

All our models are fine-tuned with a batch size of 96 and a learning rate of 7×10^{-6} for 500K steps on a single TPU-V2. For TCT-ColBERT, there are two steps in our training procedure: (1) fine-tune $\phi_{\hat{\theta}; \text{MaxSim}}$ as our teacher model, (2) freeze $\phi_{\hat{\theta}; \text{MaxSim}}$ and distill knowledge into our student model ϕ_{θ} . We keep all the hyperparameter settings the same but adjust the temperature $\tau = 0.25$ for knowledge distillation at the second step. For all our models, including the baseline, we initialize the student model using the fine-tuned weights of the teacher model in the first step. We limit the input tokens to 32 (150) for queries (passages). To evaluate effectiveness, we encode all passages in the corpus and conduct a brute-force search over the vector representations.

Our main results, including paired t -test for significance testing, are shown in Table 3.2. In addition to presenting the effectiveness of the student models, we also report the effectiveness of the teacher models used in the knowledge distillation methods.²

First, we observe that pairwise knowledge distillation methods show significant improvements over the baseline, indicating that information from BM25 negatives cannot be fully exploited without teacher models. Second, although KD-T2 improves the bi-encoder’s effective-

²We report our trained ColBERT’s accuracy by reranking the top-1000 candidates provided officially.

Table 3.3: Passage retrieval results with hard negative training. All our implemented models are labeled with a number and superscripts represent significant improvements over the labeled model (paired t -test, $p < 0.05$). Note that Bonferroni correction is not applied, so the significance tests may risk overconfidence.

Model	Training cost		MARCO Dev		DL19
	# Index Refresh	Batch Size	MRR@10	R@1K	NDCG@10
ANCE [Xiong et al., 2021]	60	32	0.330	0.959	0.648
LTRe [Zhan et al., 2020a]	60	32	0.341	0.962	0.675
SEED-Encoder [Lu et al., 2021]	≥ 10 (est.)	-	0.339	0.961	-
RocketQA [Qu et al., 2020]	1	128	0.310	-	-
RocketQA [Qu et al., 2020]	1	4096	0.364	-	-
(1) TCT-ColBERT	0	96	0.344	0.967	0.685
(2) w/ HN	1	96	0.237	0.929	0.543
(3) w/ TCT HN	1	96	0.354 ^{1,2}	0.971 ^{1,2}	0.705 ²
(4) w/ TCT HN+	1	96	0.359 ^{1,2}	0.970 ¹	0.719 ^{1,2}

ness over KD-T1, it is not consistently better than KD-ColBERT in terms of the students’ effectiveness. We suspect that they have comparable capabilities to discriminate most paired passages (BM25 negative vs. positive samples), i.e., ColBERT is good enough to guide bi-encoder student models to discriminate them. On the other hand, our TCT-ColBERT model, which uses only one teacher model and adds only 33% more training time over the baseline, yields the best effectiveness, demonstrating the advantages of our proposed in-batch knowledge distillation — exhaustive exploitation of all query–document combinations in a mini-batch.

Training with Hard Negatives. We evaluate TCT-ColBERT when training with hard negatives (HNs). We compare our model to four competitive approaches:

1. ANCE [Xiong et al., 2021] is the most representative work, which proposes asynchronous index refreshes to mine hard negatives. The model is trained for 600K steps with index refreshes every 10K steps. ANCE uses RoBERTa-base as its backbone.
2. LTRe [Zhan et al., 2020a] further improves from an ANCE checkpoint by adding more training steps with the same hard negative mining approach; thus, the computation cost of index refreshes from ANCE cannot be neglected. LTRe also use RoBERTa-base as its backbone.
3. SEED-Encoder [Lu et al., 2021] leverages a pre-training strategy to enhance the capability of the bi-encoder, which is further fine-tuned with HNs using asynchronous index refreshes.

4. RocketQA [Qu et al., 2020] trains a bi-encoder model using hard negatives denoised by a cross-encoder, ERNIE-2.0-Large [Sun et al., 2019]. It further demonstrates that training bi-encoders with many in-batch negatives (batch size up to 4096) significantly improves ranking effectiveness; however, this approach is computationally expensive (the authors report using $8 \times \text{V100}$ GPUs for training). To the best of our knowledge, RocketQA represents the state of the art in single-vector bi-encoders for dense retrieval. For a more fair comparison, we also report the ranking effectiveness of their model trained with a smaller batch size of 128.

For all the approaches above, we directly copy the reported effectiveness from the original papers.

For our TCT-ColBERT model, following the settings of the above approaches, we first use our TCT-ColBERT model trained on BM25 negatives as a warm-up starting point and index all 8.8M MARCO passages. Using the warmed-up index, we retrieve top-200 passages for each training query and randomly sample (with replacement) hard negatives from the 200 candidates to form our training data. Note that due to resource limitations we do not conduct experiments with asynchronous index refreshes since multiple V100 GPUs are required for such a model training scheme.³ In this experiment, all the hyperparameter settings are the same as the ones in the BM25 negative training, except for training steps, which is set to 100K for both student and teacher training.

Table 3.3 reports the results of our experiments with hard negative training. First, we observe that our TCT-ColBERT model trained with BM25 negatives marginally outperforms the other models trained with HNs, except for RocketQA. Comparing the different training strategies (second main block of the table), we see that the ranking effectiveness of TCT-ColBERT (HN) degrades when training on hard negatives without the guide of a teacher. This is consistent with the findings of Qu et al. [2020] that hard negatives contain noisy information (i.e., some hard negatives may actually be relevant). Also, Xiong et al. [2021] show that training bi-encoders with hard negatives can be unstable: hard negatives benefit ranking effectiveness only under certain hyperparameter settings.

In contrast, hard negative training using ColBERT’s in-batch knowledge distillation further boosts ranking effectiveness, especially when our teacher (ColBERT) is trained with the same hard negative samples beforehand. It is also worth noting that our TCT-ColBERT (w/ TCT HN+) with batch size 96 yields competitive ranking effectiveness compared to RocketQA (the current state of the art), which uses batch size 4096. These results demonstrate the advantages of our TCT design: our approach effectively exploits hard negatives in a computationally efficient manner (i.e., without the need for large batch sizes or periodic index refreshes).

³Re-encoding the entire corpus takes ~ 10 hours on one GPU.

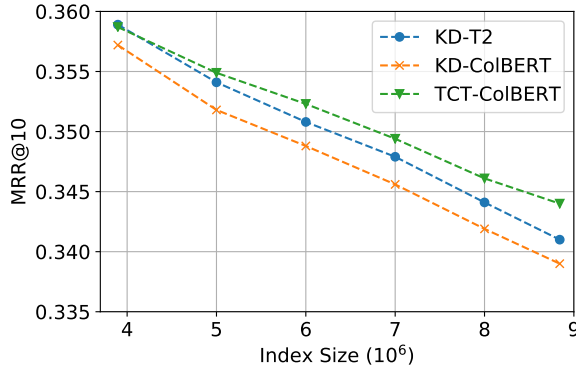
Table 3.4: Document retrieval results using the FirstP approach. All our implemented models are labeled with a number and superscripts represent significant improvements over the labeled model (paired t -test, $p < 0.05$). Note that Bonferroni correction is not applied, so the significance tests may risk overconfidence.

Model	MARCO Dev	DL19
	MRR@100	NDCG@10
ANCE [Xiong et al., 2021]	0.368	0.614
LTRe [Zhan et al., 2020a]	-	0.634
SEED-Encoder [Lu et al., 2021]	0.394	-
(1) TCT-ColBERT	0.339	0.573
(2) w/ TCT HN+	0.392 ¹	0.613
(3) w/ 2× TCT HN+	0.418 ^{1,2}	0.650 ^{1,2}

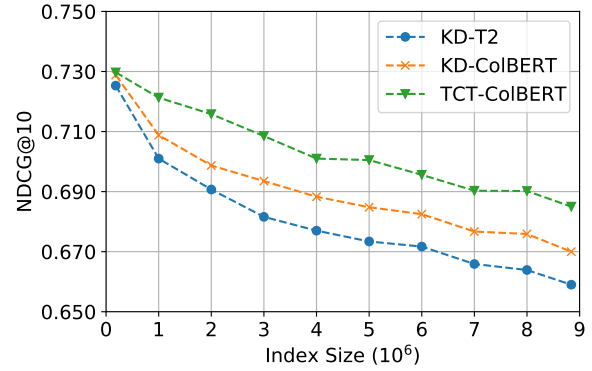
3.4.2 Document Retrieval

Ranking effectiveness is reported in Table 3.4. First, we observe that TCT-ColBERT (our warmed-up checkpoint) performs far worse than other approaches to document retrieval using the FirstP method. This may be due to the fact that FirstP document retrieval is very different from passage retrieval, making zero-shot transfer ineffective. After applying HN training on both teacher and student models (condition 2), the ranking effectiveness increases significantly. In addition, we find that another iteration of training with an index refresh (condition 3) further improves ranking effectiveness. To sum up, in the document ranking task, TCT-ColBERT yields competitive effectiveness with a one-time index refresh and outperforms other computationally expensive methods with one additional index refresh.

Analysis To understand why TCT-ColBERT yields better results, we study the models’ retrieval effectiveness against carefully selected distractors. We start with a small synthetic corpus composed of the relevant passages and the top-1000 BM25 candidates of the 6980 (43) queries from MARCO Dev (TREC-DL ’19). To increase the corpus size, we gradually add passages uniformly sampled from the corpus without replacement. From Figure 3.2, we see that the three knowledge distillation models exhibit nearly the same effectiveness when the corpus only contains BM25 candidates. This shows that the bi-encoders learn to discriminate relevant passages from the BM25 negative samples well. However, as the index size increases, TCT-ColBERT demonstrates better ranking effectiveness than the other pairwise knowledge distillation methods, indicating that the learned representations are more robust. We attribute this robustness



(a) MARCO Dev



(b) TREC-DL '19

Figure 3.2: Passage retrieval effectiveness on a synthetic corpus comprising relevant passages and BM25 results as additional “distractors” randomly sampled from the corpus are added.

against “distractors” to the enriched information from in-batch knowledge distillation, where we are able to exploit all in-batch query–document combinations.

3.5 Summary

In this chapter, we propose a novel teacher–student knowledge distillation approach using tightly coupled bi-encoders, which enables exhaustive use of query–passage combinations in each mini-batch. More importantly, knowledge distillation with a bi-encoder teacher requires less computation than using a cross-encoder teacher. We demonstrate that our model, TCT-ColBERT, requires fewer computational resources while achieving competitive retrieval effectiveness compared to state-of-the-art dense retrieval models.

Chapter 4

Training a Robust Dense Retrieval Model with Diverse Augmentation

4.1 Overview

In addition to the approaches introduced in Chapter 3, various methods have been proposed to improve the effectiveness of the single-vector bi-encoder, such as pre-training [Lee et al., 2019, Chang et al., 2020], query augmentation [Oguz et al., 2022], and distillation [Lin et al., 2021c, Hofstätter et al., 2021, Ren et al., 2021, Zeng et al., 2022]. For example, pre-training on the MS MARCO corpus improves accuracy in the fully supervised setting, while leveraging other corpora can enhance transfer in the zero-shot setting. However, we observe that an improvement in one setting is achieved only at the expense of the other. Figure 4.1 plots existing state-of-the-art dense retrieval models with respect to their effectiveness on these two axes, revealing a clear tradeoff between supervised and zero-shot effectiveness (the blue line). The only exception is GTR-XXL [Ni et al., 2022], which breaks the effectiveness tradeoff at the expense of efficiency (i.e., query encoding), leveraging very large pre-trained models with 40 times more parameters. This effectiveness tradeoff has led some to hypothesize that we have fully exploited the capacity of BERT-base-sized dense retrieval models [Ni et al., 2022] and to explore how to cleverly increase model parameters without sacrificing retrieval efficiency. For example, recent work [Wang et al., 2022, Dai et al., 2023] proposes training one expert dense retriever for each specific scenario, which results in slow adaptation to real-world applications.

In this chapter, we challenge this hypothesis and show that a generalizable dense retrieval model can indeed be trained to achieve state-of-the-art effectiveness in both supervised and

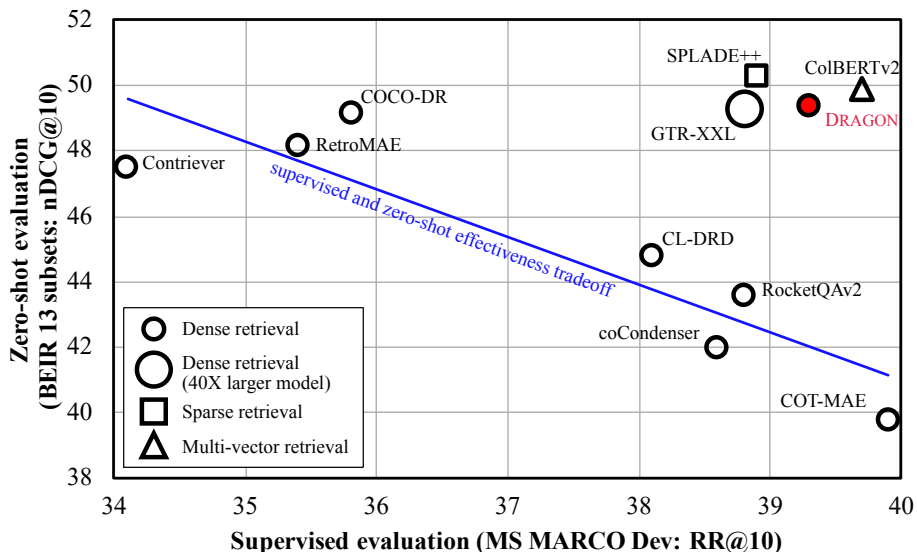


Figure 4.1: Supervised versus zero-shot effectiveness comparison among existing state-of-the-art retrievers. All models use a BERT-base-sized (110M parameters) backbone except for GTR-XXL (4.8B parameters).

zero-shot evaluations without increasing model size. To this end, we first investigate the important factors contributing to the recent progress in dense retrieval. For example, dense retrieval seems to gain zero-shot transfer capability from pre-training on large-scale and diverse training queries [Izacard et al., 2022, Yu et al., 2022], while knowledge distillation can improve the quality of supervision by automatically identifying relevant passages that are not labeled by humans [Ren et al., 2021, Zeng et al., 2022]. To better understand these approaches, we devise a unified framework of data augmentation for contrastive learning. Within this framework, prior work can be viewed as data augmentation with different strategies for query augmentation and relevance label augmentation, as shown in Table 4.1. The data augmentation framework also helps us design comprehensive studies for improved dense retrieval training.

Guided by a detailed empirical exploration within the space of our data augmentation framework, we find the following: (1) For relevance label augmentation, we identify that the key to training a generalizable dense retriever is to create diverse relevance labels for each query. To achieve this, we use multiple retrievers instead of a strong cross-encoder. (2) With such diverse relevance labels, dense retrievers can be trained effectively using inexpensive and large-scale augmented queries (e.g., cropped sentences from a corpus) instead of the more expensive neural generative queries. This finding opens the door to the future development of cheap yet useful training data at scale for dense retrieval. Finally, we find that it is suboptimal for a dense retriever

Table 4.1: Categorization of existing dense retrieval models by their approaches to data augmentation.

Model	Qry Aug.	Label Aug.	Corpus
RocketQAv2 [Ren et al., 2021] CL-DRD [Zeng et al., 2022]	$\times$	CE	MARCO
coCondenser [Gao and Callan, 2022] Contriever [Izacard et al., 2022] COCO-DR [Yu et al., 2022]	cropping	$\times$	MARCO Wiki+ CCnet BEIR
GPL [Wang et al., 2022] PTR [Dai et al., 2023]	GenQ	$\times$	BEIR
DRAGON	cropping+GenQ	retrievers	MARCO

to learn diverse relevance labels from multiple retrievers directly. Therefore, we propose a simple strategy to progressively augment relevance labels, which guides dense retrievers in learning diverse relevance signals more effectively.

4.2 A Unified Framework of Improved Dense Retrieval Training: Data Augmentation

Data augmentation for contrastive learning has been widely used in many machine learning tasks [Chen et al., 2020, Thakur et al., 2021a]. In fact, many recent approaches to train better dense retrieval, such as knowledge distillation, contrastive pre-training and pseudo query generation (GenQ), can be considered data augmentation with different recipes respectively listed in the first three main rows of Table 4.1. We compare the data augmentation recipes from the perspectives of query and relevance label augmentation.

1. Query augmentation: There are two common automatic approaches to increase the size of training queries from a given corpus, sentence cropping and pseudo query generation. The former can easily scale up query size without any expensive computation, which is used by the models for contrastive pre-training [the second section of Table 4.1; Gao and Callan, 2022, Izacard et al., 2022, Wu et al., 2023]. The latter generates quality but more expensive human-like queries using large language models for dense retrieval pre-training [Oguz et al., 2022] or domain adaptation [the third section of Table 4.1; Wang et al., 2022, Dai et al., 2023].

2. Relevance label augmentation. The aforementioned approaches to query augmentation often assume that the (or part of the) original document is relevant to the augmented queries, which may not be true and only provides a single view of relevance labeling. The recent work [the first section of Table 4.1; Ren et al., 2021, Zeng et al., 2022] improve dense retrieval training with the positive passages predicted by cross encoders. These knowledge distillation approaches further improve training data quality through label augmentation, inspiring us to conduct relevance label augmentation on the augmented queries (i.e., cropped sentences and GenQ).

4.3 Pilot Studies on Data Augmentation

Settings for Empirical Studies. We introduce some basic experimental settings to facilitate the presentation of our empirical studies on data augmentation. More detailed settings can be found in Section 4.4. Following the experiment settings in the thesis aforementioned in Section 2.5, we consider MS MARCO [Bajaj et al., 2016] as supervised data and BEIR datasets for zero-shot evaluations. Thus, we use the 8.8 million MS MARCO passage corpus to conduct data augmentation and evaluate our trained models on MS MARCO Dev, consisting of 6980 queries from the development set with one relevant passage per query on average. We report MRR@10 (abbreviated as RR@10) and Recall@1000 (R@1K) as the evaluation metrics. For zero-shot evaluations, we use BEIR [Thakur et al., 2021b], consisting of 18 IR datasets spanning diverse domains and tasks including retrieval, question answering, fact checking, question paraphrasing, and citation prediction. We report the averaged nDCG@10 over 13 public BEIR datasets, named BEIR-13, making the numbers comparable to most existing approaches [Formal et al., 2021a, Santhanam et al., 2022b].¹

We first discuss the exploration space of data augmentation based on the framework in Section 4.2 and then conduct empirical studies on how to better train a dense retriever. Based on the empirical studies, we propose our data augmentation recipe to train DRAGON, a **D**ense **R**etriever with diverse **A**uGmentation**ON**.

4.3.1 The Exploration Space of Data Augmentation

1. Query augmentation: Following the discussion in Section 4.2, we consider the two common approaches to automatic query augmentation. Specifically, for sentence cropping, following Chen et al. [2022], we use the collection of 28 million sentences from the

¹CQADupStack, Robust04, Signal-1M, TREC-NEWS, BioASQ are excluded

MS MARCO corpus consisting of 8.8 million passages. As for pseudo query generation, we use the 28 million synthetic queries sampled from the query pool generated by doct5query [Nogueira and Lin, 2019]. In addition, we also consider augmenting the type of queries by mixing cropped sentences and synthetic queries.

2. Label augmentation with diverse supervisions: Although cross encoder (CE) is known to create relevance labels with strong supervision, we hypothesize that CE still cannot capture diverse matching signals between text pairs. A query is often relevant to many documents from different perspectives (e.g., semantic or lexical matching), which cannot capture by a single labeling scheme (a strong model or even human). In this work, we seek multiple sources of supervisions from existing sparse, dense and multi-vector retrievers, which are more efficient than CE and suitable for labeling a large number of queries (see discussion in Section 4.6.1).

4.3.2 Training with Diverse Supervisions

We have introduced our searching space for query and label augmentation (with diverse supervisions); however, training a dense retriever on such augmented data is not trivial. First, how can we create training data using a teacher from any augmented queries (i.e., cropped sentences or pseudo generative queries)? Second, with the training data sets created from multiple teachers, how can we train a dense retriever to digest the multiple supervisions?

Formally speaking, given N teachers, for each augmented query q , we retrieve N ranked lists (i.e., $\mathcal{P}_q^1, \mathcal{P}_q^2, \dots, \mathcal{P}_q^N$ with each list has K passages) from the corpus with the respective teachers. We consider the ranked list $\mathcal{P}_q^n$ from the n -th teacher a source of supervision since the top- k and last- k' passages in $\mathcal{P}_q^n$ contain the teacher’s view on what is relevant and less relevant for the given query. We then discuss possible strategies to train a dense retriever with diverse supervisions.

1. Fused supervision: An intuitive strategy is to fuse the multiple sources into a single high-quality supervision, which dense retrievers can learn from. For the augmented query q , we conduct linear score fusion in Eq. (2.9) on the N ranked lists to form a new ranked list $\mathcal{F}_q$ as a fused supervision.
2. Uniform supervision: Another simple strategy is to provide a dense retriever with equal exposures to multiple sources of supervisions. Specifically, given a query, we uniformly sample a source of supervision; i.e., a ranked list $\mathcal{P}_q^n$, where $n \sim \mathcal{U}(1, N)$. This approach naturally encourages the positive samples appearing in more ranked lists to be sampled and

vice versa. The advantage is that fusion weight tuning is not required. Furthermore, models can see diverse supervisions from different teachers in contrast to fused supervision, which may be dominated by a single strong teacher.

3. **Progressive supervision:** The previous two approaches directly give models supervisions from multiple teachers at once; however, learning directly from the mixture of supervision is challenging, especially for dense retrieval models which compute text matching with simple dot product. Inspired by the success of curriculum learning [Zeng et al., 2022], we propose an approach to *progressive label augmentation* to guide dense retrieval training with progressively more challenging supervision. Specifically, we train our models with uniform supervision for N iterations and at each iteration, we augment relevance label using additional teacher, as illustrated in Figure 4.2; i.e., at iteration $T \leq N$, we uniformly sample a source of supervision, $\mathcal{P}_q^n$, where $n \sim \mathcal{U}(1, T)$. A key factor of this approach is how to arrange the order for easy-to-hard supervisions; namely, the trajectory of progressive supervision.

With any aforementioned strategy to obtain diverse supervisions, we train our dense retrievers using contrastive loss in Eq. (2.6). Specifically, given a query q , we first obtain a source of supervision either from sampling ($\mathcal{P}_q^n$) or fusion ($\mathcal{F}_q$); then, we randomly sample a positive and hard negative from the top 10 passages and top 46–50 passages, respectively to form a triplet. The sampling scheme has been empirically proved to well preserve the supervised signal from a single teacher [Chen et al., 2022] (also see our study in Section 4.6.1). In this work, we further extend the sampling scheme to obtain diverse supervisions from multiple teachers.

4.3.3 Empirical Studies

Strategies to Obtain Diverse Supervisions. We first conduct empirical studies on how to better train a dense retriever in a simplified setting by using the MS MARCO cropped sentences as augmented queries and obtain supervised labels using three teachers with diverse relevance score computation: uniCOIL (sparse), Contriever (dense) and ColBERTv2 (multi-vector). We report the models trained with single (columns 0–2) and multiple (columns 3–5) sources of supervisions for 20 epochs and 60 epochs, respectively. For progressive supervision, we follow the supervision trajectory: uniCOIL $\rightarrow$ Contriever $\rightarrow$ ColBERTv2 with 20 epochs for each of the three iterations ($N = 3$). Note that for fused supervision, we use MS MARCO Dev queries to tune and obtain the best hyperparameters to create fusion list.

The results are tabulated in Table 4.2. We observe that when learning from a single supervision (columns 0–2), there is a tradeoff between supervised and zero-shot retrieval effectiveness.

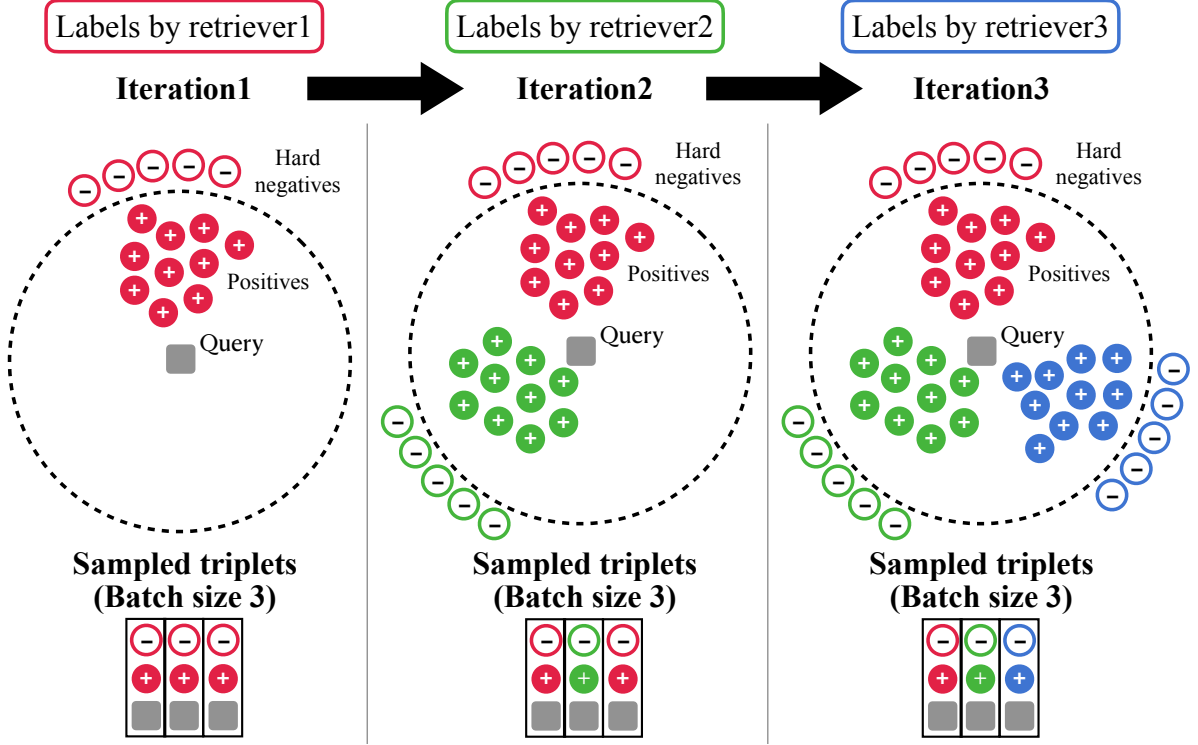


Figure 4.2: Illustration of progressive label augmentation. For each iteration of training, additional relevance labels from a teacher are augmented in the training data. By contrast, uniform supervision directly exposes models to all the supervisions (as in iteration 3) in the beginning.

Learning from the fusion list only sees a slight improvement over supervised evaluation while no improvement observes in zero-shot evaluations (columns 0–2 vs 3). By contrast, the model sees notable improvements in zero-shot evaluations when trained with uniform supervision (columns 0–3 vs 4), indicating that learning from the diverse relevance labels from multiple retrievers separately rather than single strong supervision (ColBERTv2 or fused supervision) is key to gain generalization capability. Finally, we observe that progressive supervision can further guide a dense retriever to gain generalization capability over uniform supervision (column 4 vs 5). Thus, we use progressive supervision in the following experiments.

Trajectory of Progressive Supervision. We then study how to better arrange the trajectories of progressive supervision in Table 4.3. We observe that different trajectories have much impact on models’ zero-shot retrieval effectiveness while a minor impact on supervised evaluation can be seen. For example, switching the sampling order between uniCOIL and Contriever results in a

Table 4.2: Strategies to obtain multiple supervisions using cropped sentences as queries.

	0	1	2	3	4	5*
Teacher	uniCOIL	Contriever	ColBERTv2	three teachers		
				fused	unif.	prog.
	effectiveness of student					
MARCO Dev	34.9	33.9	36.4	36.7	36.9	36.6
BEIR-13	46.7	47.0	46.3	46.6	47.7	49.3
	effectiveness of teacher					
MARCO Dev	35.1	34.1	39.7	40.0	-	-
BEIR-13	$-\triangle$	47.5	49.9	$-\triangle$	-	-

* The condition corresponds to row 0 in Table 4.3.

$\triangle$ We do not evaluate uniCOIL on BEIR due to its requirement of expensive document expansion from corpus.

degrade of 1 point on the averaged nDCG@10 over BEIR-13 (row 0 vs 1) while reversing the whole trajectory leads to a degrade with more than 1.5 points (row 0 vs 3). This observation reflects an intuition that the retrievers with better generalization capability may capture more complex matching signal between text pairs (ColBERTv2 shows better generalization capability than the other two teachers); thus, their relevance labels should be augmented at a later stage of model training. Finally, in row 3, we follow the trajectory in row 0 but only use ColBERTv2 as the only source of supervision instead of obtaining uniform supervision from the three teachers at the last iteration. This change results in worse zero-shot retrieval effectiveness, indicating that learning from diverse supervisions is key to training a generalizable dense retriever.

Query Augmentation. We study the impacts of query augmentation on models’ effectiveness under the scenario where supervision is given, which has not been studied so far in the field of dense retrieval. With the best trajectory of progressive supervision, we compare models trained using cropped sentences (rectangles), generative queries (GenQ; circles), their mixture (triangles) and human queries (cross) in Figure 4.3. We observe that query size is the key to successful training. Although training with limited (0.8M) cropped sentences cannot perform well in MS MARCO dataset, scaling up the size (28M) sees significant improvement over the model trained on 0.8M human queries. Similarly, in Figure 4.3 (b), model’s generalization capability shows a huge jump when scaling up query size. While surprisingly, cropped sentences can help dense

Table 4.3: Study on trajectory of progressive supervision using cropped sentences as queries.

Progressive supervision	MARCO Dev	BEIR-13
trajectories	RR@10	nDCG@10
(0) uniCOIL→ Contriever → ColBERTv2	36.6	49.3
(1) Contriever → uniCOIL→ ColBERTv2	36.7	48.4
(2) ColBERTv2→ Contriever→ uniCOIL	36.4	47.7
(3) uniCOIL→ Contriever → ColBERTv2*	36.8	47.4

* ColBERTv2 is the only teacher at the last (3rd) iteration.

retrievers to gain more generalization capability than human-like GenQ, a mixture of cropped sentences and GenQ yields strong retrieval effectiveness in supervised and zero-shot evaluations, when the query size is not large enough (0.8–8M).

4.4 Methodology: DRAGON

With the empirical studies on dense retrieval training, we then propose the final recipe to train our DRAGON. We train DRAGON for 20 epochs (around 130K steps) at each iteration, with the trajectory of progressive supervision: uniCOIL²→ Contriever³→ GTR-XXL⁴→ ColBERTv2⁵→ SPLADE++⁶. This trajectory is based on models’ retrieval effectiveness on BEIR with the intuition gained from Section 4.3.3 that a more generalizable model creates relevance labels with more complex matching signals. For query augmentation, we mix half of cropped sentences and synthetic queries as training queries. Note that we do not further fine-tune our models on the MS MARCO training queries. In addition, we train other three DRAGON variants. DRAGON-S and DRAGON-Q only use cropped sentences and synthetic queries, respectively. As for DRAGON +, we follow the same training procedure of DRAGON but switch the initialization from BERT to the masked auto-encoding pre-trained model, RetroMAE.⁷

²<https://huggingface.co/castorini/unicoil-msmarco-passage>

³<https://huggingface.co/facebook/contriever-msmarco>

⁴<https://huggingface.co/sentence-transformers/gtr-t5-xxl>

⁵<https://github.com/stanford-futuredata/ColBERT>

⁶http://download-de.europe.naverlabs.com/Splade_Release_Jan22/splade_distil_CoCodenser_medium.tar.gz

⁷<https://huggingface.co/Shitao/RetroMAE>

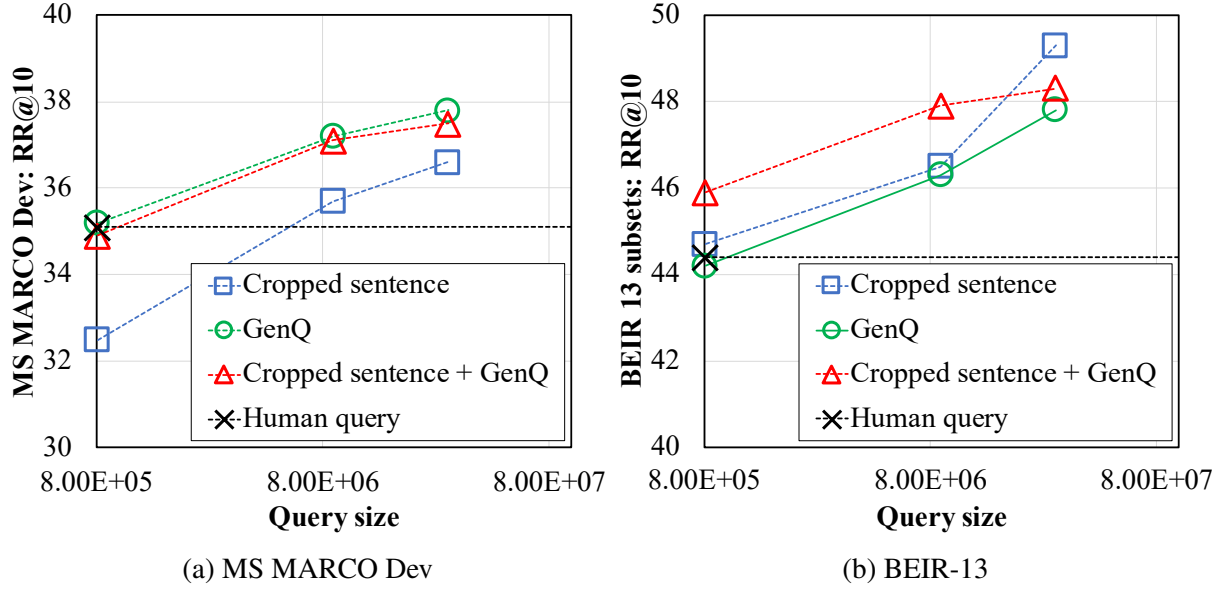


Figure 4.3: Impacts of query augmentation.

Implementation Details. We train DRAGONS initialized from bert-base-uncased on 32 A100 GPUs (40GB) with a per-GPU batch size of 64 and a learning rate of $3e - 5$. Each batch includes an augmented query with its positives and hard negatives. The statistics of our augmented training queries and average numbers of relevance labels are listed in Table 4.4. Following Karpukhin et al. [2020], we use asymmetric dual encoder with two distinctly parameterized encoders and leverage in-batch negative mining. Note that symmetric dual encoder shows poor generalization capability in our initial experiments. We set the maximum query and passage lengths to 32 and 128 for MS MARCO training and evaluation. For BEIR evaluation, we set maximum input lengths to 512.

4.5 Experimental Setups

4.5.1 Datasets

Following Section 2.5, we use MS MARCO Dev, TREC DL 19 and 20 as the supervised evaluation datasets and use BEIR-13 as the zero-shot evaluation datasets. Since we use BEIR-13 to study the impacts of different training strategies on models’ generalization capability, we further

Table 4.4: MS MARCO and our augmented training queries statistics.

	number	Avg. # tokens	Avg. # rel
passages in corpus	8,841,823	78.8	na
training queries (not used)	532,761	8.2	1.0
augmented training queries			
cropped sentences	28,545,938	24.4	23.1
generative queries	28,545,938	8.0	24.7

report models’ effectiveness on all the 18 datasets in BEIR [Thakur et al., 2021b]. In addition, we use LoTTE [Santhanam et al., 2022b] consisting of questions and answers posted on Stack-Exchange with five topics including writing, recreation, science, technology, and lifestyle. We evaluate models’ retrieval effectiveness in the pooled setting, where the passages and queries from the five topics are aggregated. Following Santhanam et al. [2022b], the retrieval effectiveness of Success@5 on search and forum queries are reported.

4.5.2 Baseline Dense Retrieval Models

We compare DRAGON with dense retrievers using the backbone of bert-base-uncased trained with advanced techniques.

1. Knowledge distillation: RocketQAv2 [Ren et al., 2021] distills knowledge from a cross encoder while CL-DRD⁸ [Zeng et al., 2022] combines curriculum learning and cross-encoder distillation. They all use cross encoders’ knowledge to augment positive relevance labels as our approach. Note that we directly copy the numbers of RocketQAv2 from Santhanam et al. [2022b].
2. Contrastive pre-training: coCondenser⁹ [Gao and Callan, 2022], Contriever¹⁰ [Izacard et al., 2022] and COCO-DR¹¹ [Yu et al., 2022] are first pre-trained on different corpus listed in Table 4.1, and then fine-tuned on MS MARCO training queries.
3. Masked auto-encoding pre-training: COT-MAE¹² [Wu et al., 2023] and RetroMAE¹³ [Xiao

⁸<https://github.com/HansiZeng/CL-DRD>

⁹<https://huggingface.co/Luyu/co-condenser-marco-retriever>

¹⁰<https://huggingface.co/facebook/contriever-msmarco>

¹¹<https://huggingface.co/OpenMatch/cocodr-base-msmarco>

¹²https://huggingface.co/caskcsg/cotmae_base_msmarco_retriever

¹³https://huggingface.co/Shitao/RetroMAE_BEIR

et al., 2022] are first pre-trained to recover polluted sentences and then fine-tuned on MS MARCO training queries. For RetroMAE, we use the variant with the best BEIR retrieval effectiveness for comparison.

4. Domain adaptation: We consider GPL [Wang et al., 2022] and Promptagator [PTR; Dai et al., 2023], which use generative models to create pseudo relevance data for each corpus in BEIR and train one expert dense retriever for each corpus. This approach requires the target corpus while training. Note that COCO-DR can also be considered domain adaptation on BEIR although it uses one model for all tasks.

Note that coCondenser and COT-MAE are fine-tuned on the “non-standard” MS MARCO passage corpus that has been augmented with title. Thus, we also conduct inference on the corpus with title for them; otherwise, we use the official MS MARCO passage corpus. In addition, we also report the retrieval effectiveness of GTR-XXL [Ni et al., 2022] and ColBERTv2 [Santhanam et al., 2022b] from their original papers and conduct retrieval for SPLADE++ using Pyserini [Lin et al., 2021a] for reference.

Table 4.5: The latency comparison of relevance label augmentation with batch inference using different teachers on MS MARCO.

Type	Model	candidates	latency (ms/q)	
		(#)	GPU	CPU
cross-encoder	miniLML6v2	1K	600	-
dense	Contriever	8.8M	< 1	-
dense	GTR-XXL	8.8M	10	-
sparse	uniCOIL	8.8M	-	84
sparse	SPLADE++	8.8M	-	144
multi-vec	ColBERTv2	8.8M	55	-

4.5.3 Efficiency Measurement of Label Augmentation

We measure the latency of label augmentation using batch retrieval on a single NVIDIA A100 40GB GPU for GPU search and 60 Intel(R) Xeon(R) Platinum 8275CL CPUs @ 3.00GHz for CPU search. For cross encoder, we conduct label augmentation by re-ranking text pairs with a batch size of 100. For dense retrieval (Contriever and GTR-XXL) and sparse retrieval, we use Faiss-GPU index and Lucene index from Pyserini [Lin et al., 2021a] with 60 threads, respectively, and search with a batch size of 100. Note that we use a batch size of 25 to encode queries

using GTR-XXL due to GPU memory constraint, which is also the main bottleneck for GTR-XXL batch retrieval. For ColBERTv2, we use the improved version of multi-vector retrieval, PLAID [Santhanam et al., 2022a], and search with a batch size of 1, which is the only option. We report the measurement in Table 4.5.

4.6 Experimental Results

Supervised Evaluations. The first main row in Table 4.6 reports models’ retrieval effectiveness on MS MARCO passage ranking dataset. We first observe that some baseline dense retrievers which perform well in MS MARCO Dev set are either pre-trained on MS MARCO corpus (coCondenser and COT-MAE) or well fine-tuned on MS MARCO training queries with cross-encoder distillation (CL-DRD and RocketQAv2). However, their retrieval effectiveness on MS MARCO Dev set is not well correlated to TREC DL queries, which have fine-grained human labels with different degrees of relevance. We hypothesize that these models are able to retrieve the most relevant passage from the corpus but cannot retrieve diverse passages with different degrees of relevance. By contrast, all the variants of DRAGON trained with diverse augmented relevance labels show consistently strong effectiveness in MS MARCO Dev and TREC DL queries.

Zero-Shot Evaluations. The second main row in Table 4.6 reports models’ zero-shot retrieval effectiveness on the BEIR datasets. We observe a reverse trend that those dense retrievers performing relatively poorly in MS MARCO Dev queries have better zero-shot retrieval effectiveness (e.g., Contriever, COCO-DR and RetroMAE). These models are pre-trained (with data augmentation) on a corpus other than MS MARCO to combat domain shift issue in dense retrieval [Xin et al., 2022, Yu et al., 2022]. On the other hand, DRAGON s trained on augmented data from MS MARCO corpus only transfer well to BEIR datasets. Furthermore, DRAGON + reaches state-of-the-art retrieval effectiveness on BEIR as the sparse retriever, SPLADE++. In addition, all the DRAGON variants outperform other dense retrievers by a large margin and compete SPLADE++ and ColBERTv2 in the LoTTE dataset. It is worth mentioning that the models trained with domain adaptation (columns 9–B) perform better than the others (columns 3–8) but still underperform DRAGON in zero-shot evaluations. Using DRAGON as the base model for domain adaptation is possible to gain retrieval effectiveness, which we leave to our future work.

A comparison between DRAGON-S and DRAGON-Q (columns C and D) shows that augmented query type has an impact on retrieval effectiveness in different datasets. DRAGON-S trained on cropped sentences surprisingly sees the highest retrieval effectiveness on BEIR but only sacrifices a bit on MS MARCO datasets. This means that cropped sentences, the cheap

Table 4.6: Comparison with existing state-of-the-art dense retrievers. Bold (underline) denotes the best (second best) effectiveness for each row among baseline dense models.

Rep type	sparse	mul-vec	dense	baseline dense										our dense			
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
	SPLADE++	ColBERTv2	GTR-XXL	CL-DRD	RocketQA _{v2}	COT-MAE	RetroMAE	coCondenser	Contriever	COCO-DR	GPL	PTR	DRAGON-S	DRAGON-Q	DRAGON	DRAGON+	
Pre-training	✓	✗	✓	✗	✗	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗	✓	
Distillation	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓	
Target Corpus [†]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	
MS MARCO (Supervised)																	
Dev (RR@10)	38.9	39.7	38.8	38.1	38.8*	39.9*	35.4	38.6*	34.1	35.8	-	-	38.1	39.1	<u>39.3</u>	39.0	
Dev (R@1K)	98.2	98.4	99.0	97.9	98.1*	98.5*	97.5	98.4*	97.9	97.9	-	-	98.3	98.8	98.5	98.6	
DL2019 (nDCG@10)	74.3	74.6	-	72.5	-	70.0*	68.8	71.5*	67.8	<u>74.1</u>	-	-	73.6	74.0	74.1	74.4	
DL2020 (nDCG@10)	71.8	75.2	-	68.3	-	67.8*	71.4	68.1*	66.1	69.7	-	-	70.0	<u>72.6</u>	72.9	72.3	
nDCG@10	BEIR (Zero-shot)																
TREC-COVID	71.1	73.8	50.1	58.4	67.5	56.1	<u>77.2</u>	71.2	59.6	78.9	70.0	72.7	73.9	73.2	74.0	75.9	
NFCorpus	34.5	33.8	34.2	31.5	29.3	32.1	30.8	32.5	32.8	35.5	<u>34.5</u>	33.4	32.2	33.0	32.9	33.9	
FiQA-2018	35.1	35.6	46.7	30.8	30.2	28.3	31.6	27.6	32.9	31.7	34.4	40.4	<u>35.6</u>	35.3	35.0	<u>35.6</u>	
ArguAna	52.1	46.3	54.0	41.3	45.1	27.8	43.3	29.9	44.6	49.3	55.7	<u>53.8</u>	51.5	45.5	48.9	46.9	
Touche-2020	24.4	26.3	25.6	20.3	24.7	21.9	23.7	19.1	23.0	23.8	25.5	26.6	<u>26.5</u>	26.0	24.9	26.3	
Quora	81.4	85.2	89.2	82.6	74.9	75.6	84.7	85.6	86.5	86.7	83.6	-	86.4	<u>87.1</u>	86.9	87.5	
SCIDOCS	15.9	15.4	16.1	14.6	13.1	13.2	15.0	13.7	<u>16.5</u>	16.0	16.9	16.3	15.9	15.0	15.4	15.9	
SciFact	69.9	69.3	66.2	62.1	56.8	60.1	65.3	61.5	67.7	70.9	67.4	62.3	67.8	67.2	67.5	<u>67.9</u>	
NQ	54.4	56.2	56.8	50.0	50.5	48.3	51.8	48.7	49.5	50.5	48.3	-	<u>53.3</u>	52.3	53.1	53.7	
HotpotQA	68.6	66.7	59.9	58.9	53.3	53.6	63.5	56.3	63.8	61.6	58.2	60.4	<u>65.6</u>	62.7	64.8	66.2	
DBPedia	44.2	44.6	40.8	38.1	35.6	35.7	39.0	36.3	41.3	39.1	38.4	36.4	40.6	41.4	<u>41.4</u>	41.7	
FEVER	79.6	78.5	74.0	73.4	67.6	50.6	77.4	49.5	75.8	75.1	75.9	76.2	<u>76.4</u>	75.1	75.8	78.1	
Climate-FEVER	22.8	17.6	26.7	20.4	18.0	14.0	23.2	14.4	23.7	21.1	<u>23.5</u>	21.4	21.8	20.3	22.2	22.7	
CQADupStack	34.1	-	39.9	32.5	-	29.7	34.7	32.0	34.5	37.0	35.7	-	<u>35.9</u>	34.4	35.2	35.4	
Robust04	45.8	-	50.6	37.7	-	30.8	44.7	35.4	<u>47.6</u>	44.3	43.7	-	46.3	45.3	47.2	47.9	
Signal-1M	29.6	-	27.3	28.2	-	21.1	26.5	28.1	19.9	27.1	27.6	-	<u>29.3</u>	28.9	30.1	<u>30.1</u>	
TREC-NEWS	39.4	-	34.6	38.0	-	26.1	<u>42.8</u>	33.7	<u>42.8</u>	40.3	42.1	-	42.1	40.0	41.6	44.4	
BioASQ	50.4	-	32.4	37.4	-	26.2	42.1	25.7	38.3	42.9	44.2	-	41.3	41.9	41.7	<u>43.3</u>	
Averaged nDCG@10																	
PTR 11 subsets	47.1	46.2	44.9	40.9	40.1	35.7	44.5	37.4	43.8	45.7	45.5	45.5	<u>46.2</u>	45.0	45.7	46.5	
BEIR-13	50.3	49.9	49.3	44.8	43.6	39.8	48.2	42.0	47.5	49.2	48.6	-	<u>49.8</u>	48.8	49.4	50.2	
Avg. all 18 datasets	47.4	-	45.8	42.0	-	36.2	45.4	38.9	44.5	46.2	45.9	-	<u>46.8</u>	45.8	46.6	47.4	
Success@5	LoTTE (Zero-shot)																
Search (pooled)	70.9	71.6	-	65.8	69.8	63.4	66.8	62.5	66.1	67.5	-	-	71.4	72.4	<u>72.6</u>	73.5	
Forum (pooled)	62.3	63.4	-	55.0	57.7	51.9	58.5	52.1	58.9	56.8	-	-	61.1	61.2	<u>61.4</u>	62.1	

[†] The approach assumes the target corpus (e.g., BEIR) is available while training.

* These numbers are not comparable due to the use of non-standard MS MARCO corpus augmented with title [Lassance and Clinchant, 2023].

query type (compared to neural generative queries), are sufficiently helpful for models to learn domain-invariant retrieval capability. By contrast, we observe that DRAGON-Q trained with human-like queries performs poorly compared to DRAGON-S on the datasets where queries are far different from human-like queries, such as ArguAna (45.5 vs 51.5) and CQADupStack (34.4

Table 4.7: Label augmentation with cross encoder (CE) using cropped sentences as queries.

	0 [†]	1	2	3
Initialization with DRAGON-S		✓	✓	
Source of supervision		all*+ CE	CE only	CE only
MARCO Dev (RR@10)	38.1	38.1	37.5	36.8
BEIR-13 (nDCG@10)	49.8	49.7	48.7	47.7

[†] Column 0 corresponds to DRAGON-S.

* “all” denotes the five teachers used for training DRAGON-S.

vs 35.9), while mixing different types of queries (DRAGON) can mitigate the issue. Finally, DRAGON +, combined with masking auto-encoding (MAE) pre-training and our approach, sees further improvement on zero-shot evaluations without sacrificing in-domain ones, indicating that MAE pre-training may be orthogonal to our approach based on contrastive learning.

To sum up, DRAGONs advance state-of-the-art zero-shot effectiveness while keeping strong effectiveness in supervised evaluation. The experimental results demonstrate that our data augmentation approaches enable dense retrievers to learn domain-invariant matching signal between text pairs as the other models with fine-grained late interaction (SPLADE++ and ColBERTv2) or 40 times larger model size (GTR-XXL).

4.6.1 Discussions

Is it necessary to augment relevance labels with a cross encoder? To answer this question, we further train DRAGON-S with the augmented relevance labels from a cross encoder (CE). Specifically, we create a ranked list of cross-encoder by first retrieving top 1000 passages with DRAGON-S and re-ranking them with the CE for each cropped sentence as a query. With the CE ranked list, we conduct another iteration (20 epochs) of training for DRAGON-S; however, we do not see retrieval effectiveness improvement in Table 4.7 (column 0 vs 1). In addition, the retrieval effectiveness becomes even worse when we further train DRAGON-S by only sampling CE ranked list instead of uniformly sampling all the six ranked lists (column 1 vs 2). Finally, we initialize from bert-base-uncased and re-train the model for three iterations (60 epochs) only with the CE ranked list.¹⁴ We observe that its effectiveness (column 3) is even worse than the models trained with the ranked lists from three retrievers (see columns 4 and 5 in Table 4.2). This result contradicts the general belief that CE provides the strongest supervision to a dense

¹⁴We do not notice effective improvement with more iterations of training both in supervised and zero-shot evaluations.

Table 4.8: Ablation on initialization using the mixture of cropped sentences and GenQ as queries.

Initialized checkpoint	MARCO Dev	BEIR-13
	RR@10	nDCG@10
(0) BERT base (DRAGON)	39.3	49.4
(1) Contriever	38.7	49.3
(2) RoBERTa base	39.4	49.9
(3) RetroMAE (DRAGON+)	39.0	50.2

retriever [Hofstätter et al., 2020, Ren et al., 2021]. Moreover, it demonstrates the effectiveness of using diverse supervisions to train a generalizable dense retriever, rather than relying on a single strong supervision. Furthermore, as shown in Table 4.5, leveraging all the retrievers to augment labels is still more efficient than a cross encoder.

Does DRAGON benefit from unsupervised pre-training? Table 4.8 compares the models trained from different checkpoint initialization. Note that for Contriever and RetroMAE, we initialize from the checkpoint with only unsupervised pre-training (without fine-tuning on MS MARCO). We observe that our approach benefits from further masked auto-encoding rather than contrastive pre-training (rows 2,3 vs 1). The result is sensible since our approach can be considered an improved version of contrastive pre-training, which appears to be orthogonal to masked auto-encoding pre-training. We leave the investigation of improving dense retrieval with generative and contrastive pre-training combined for future research.

Can we use the soft labels from multiple teachers? In the literature, using the relevance scores from a teacher as soft labels is a standard of knowledge distillation [Lin et al., 2021c, Hofstätter et al., 2021]. However, in our study, even when training with uniform supervision from a sparse and dense retriever (i.e., uniCOIL and Contriever), it is challenging to normalize their relevance scores and create universal soft labels, yielding significant supervised and zero-shot effectiveness drops. We suspect that dense and sparse retrievers have many different views on relevance score computation; thus, it is even harder for a dense retriever to learn the score distributions from the different teachers.

Impacts of Top- k Positive Sampling. In DRAGON training, our sampling scheme treats top 10 passages from each teacher ranked list as positives and top 45–50 as negatives. We further conduct experiment to study the impact of the positive sampling scheme. Following the experiment setups in Section 4.3.3, we use the sentences cropped from MS MARCO corpus as augmented queries and we conduct progressive label augmentation using top- k passages as positive. The

Table 4.9: Ablation on progressive label augmentation from top- k passages using cropped sentences as queries with trajectory: uniCOIL $\rightarrow$ Contriever $\rightarrow$ ColBERTv2.

top- k positives	1	5	10
MARCO Dev (RR@10)	33.1	36.4	36.6
BEIR-13 (nDCG@10)	42.4	48.0	49.3

Table 4.10: Uniform and progressive supervision effectiveness comparison at each training iteration. The models are trained using cropped sentences as queries.

Teacher	uniform			progressive		
	1st $\rightarrow$	2nd $\rightarrow$	3rd iteration	1st $\rightarrow$	2nd $\rightarrow$	3rd iteration
uniCOIL	✓	✓	✓	✓	✓	✓
Contriever	✓	✓	✓	✗	✓	✓
ColBERTv2	✓	✓	✓	✗	✗	✓
MARCO Dev	36.2	37.0	36.9	34.9	35.8	36.6
BEIR-13	46.6	47.4	47.6	46.7	48.6	49.3
effectiveness of teacher						
MARCO Dev	39.1	39.1	39.1	35.1	36.5	39.1
diversity of teacher						
Avg. # rel	17.5	17.5	17.5	10.0	14.9	17.5

results are tabulated in Table 4.9. We observe that treating top-10 passages from each teacher as positives yields the best supervised and zero-shot effectiveness. On the other hand, using only the top passage as positive results in significant effectiveness drop. This result indicates that the top passage labeled by a teacher cannot transfer its knowledge well to a student. This result is similar to the observation from Chen et al. [2022].

An Intuition Behind Uniform and Progressive Supervisions As shown in Section 4.3, uniform supervision provides good supervision without fusion weight tuning as fused supervision. Intuitively, a positive retrieved by more teachers has a higher probability to be sampled and may be more relevant to a query. To provide a sense of why uniform supervision works, we estimate the accuracy of supervision by computing the probability of each positive sampled under uniform supervision, and rank the positives according to the simulated probability. For example, at the 3rd iteration of progressive training, given a query, a positive passage is labeled positive by all the three teachers, the probability of the passages being sampled is $\frac{1}{3} \cdot (\frac{1}{k} + \frac{1}{k} + \frac{1}{k}) = \frac{1}{k}$. In our experiments, each teacher labels the top 10 ($k = 10$) retrieved passages as positives in

Passage: The Manhattan Project and its atomic bomb helped bring an end to World War II. Its legacy of peaceful uses of atomic energy continues to have an impact on history and science.				
Query Augmentation		Augmented rel (#)	Example of augmented rel	
Cropping	Q1	The Manhattan Project and its atomic bomb helped bring an end to World War II	30	The Manhattan project was a secret research and development project of the U.S to develop the atomic bomb. Its success granted the U.S the bombs that ended the war with Japan as well as ushering the country into the atomic era.
	Q2	Its legacy of peaceful uses of atomic energy continues to have an impact on history and science.	30	An early nuclear power plant that used atomic energy to generate electricity. The Atomic Age, also known as the Atomic Era, is the period of history following the detonation of the first nuclear (atomic) bomb, ...
GenQ	Q1	what were a major contributions to the manhattan effort	26	The Manhattan Project was an effort during World War II in the United States to develop the first nuclear weapon.
	Q2	what impact did the manhattan project have on history	26	The Manhattan Project , which included some of history's greatest scientific minds, lead to the end of the war against the Japanese. But was it worth the environmental and financial costs? This massive site provides loads of ...

Figure 4.4: Examples of augmented queries and relevance labels from a passage. Augmented rel (#) denotes the number of unique relevant passages labeled by all our five teachers.

our labeling scheme. Note that, in the case where multiple positives have equal probability, we further rank them according to their sum of reciprocal rank. For instance, if the two passages (e.g., p_1 and p_2) are retrieved by all the three teachers; then, we further rank them according to their scores $\frac{1}{r_{11}} + \frac{1}{r_{12}} + \frac{1}{r_{13}}$ and $\frac{1}{r_{21}} + \frac{1}{r_{22}} + \frac{1}{r_{23}}$, where r_{mn} denotes the rank of the passage p_m by the n -th teacher. In addition, we also estimate the diversity of supervision by computing the number of positive passages in union sets from the sources of supervisions.

Why sentence cropping yields a generalizable dense retriever? Figure 4.4 showcases the augmented queries by sentence cropping and neural generation and their respectively augmented relevant passages other than the original passages. We observe two main differences between the cropped sentences and generative queries. Cropped sentences provide diverse queries from a passage; i.e., the two cropped sentences in Figure 4.4 include slightly different topics (Manhattan Project and atomic energy). By contrast, all generative queries surround the same main topic, Manhattan Project, about the original passages. Second, the cropped sentences have more unique augmented relevant passages than generative queries. This is maybe because a cropped sentence, containing more information (keywords), is more challenging than a generative human-like query. Thus, teachers show more disagreement between each other on cropped sentences. We hypothesize that a dense retriever trained on cropped sentences can capture more diverse

supervised signals from multiple teachers than generative queries. This explains the reason why DRAGON-S shows better generalization capability than DRAGON-Q.

4.7 Summary

In this chapter, we present a simple yet novel data augmentation approach to enhance the robustness of dense retrieval models across various domains and tasks, without the need to crawl large-scale training data from the web. We propose a unified framework for data augmentation to better understand the recent progress in training dense retrieval models. Based on this framework, we extensively study how to improve dense retrieval training through query and relevance label augmentation. Our experiments reveal some insights into training a dense retrieval model, which contradict the common wisdom that a cross-encoder is the most effective teacher and human-like queries are the most suitable training data for dense retrieval. We then propose a diverse data augmentation recipe, including query augmentation with a mixture of sentence cropping and generative queries, as well as progressive relevance label augmentation with multiple teachers. Using this proposed recipe, we are the first to demonstrate that a single BERT-base-sized model can achieve state-of-the-art effectiveness in both supervised and zero-shot retrieval tasks.

Chapter 5

A Dense Representation Framework for Semantic and Lexical Matching

5.1 Overview

Semantic matching through dense representations [Reimers and Gurevych, 2019, Chang et al., 2020, Karpukhin et al., 2020] forms a large and successful class of models.¹ These dense semantic representations transform the relevance matching problem into nearest neighbor search in a semantic space, addressing vocabulary and semantic mismatches in ways that traditional lexical matching approaches (e.g., BM25) cannot. Subsequent work has further improved these models through advanced training techniques [Lin et al., 2021c, Hofstätter et al., 2021, Xiong et al., 2021, Zhan et al., 2021b, Gao and Callan, 2021, Izacard et al., 2022, Qu et al., 2021]. However, as Sciavolino et al. [2021] show, dense semantic representations still fail in some easy cases, and it remains challenging to interpret why they sometimes perform poorly. Additionally, Thakur et al. [2021b] demonstrate that many existing dense retrievers still fall short in terms of generalization capability across different domains.

There is evidence [Luan et al., 2021, Gao et al., 2021b, Lin and Ma, 2021] that lexical matching compensates for the weaknesses of semantic matching; these studies further propose hybrid retrieval techniques that fuse lexical and semantic representations. However, in practice, lexical and semantic matching are executed in very different ways: typically, lexical matching is conducted using inverted indexes, such as in Lucene, while semantic matching is treated as nearest

¹In the literature, these are often referred to simply as dense retrieval models. However, we explicitly call these dense *semantic* representations because we show that dense *lexical* representations exist as a separate class of models.

neighbor search using, for example, HNSW indexes in Faiss [Johnson et al., 2021]. This means that a hybrid retrieval system requires two separate “software stacks”, running completely distinct retrieval operations in parallel before their outputs are post-processed to generate a final ranking (e.g., through a linear combination of scores). Such a design makes real-world deployments more complicated than necessary, as, for example, the two separate indexes need to be maintained and kept in sync.

Recently, another thread of work uses bi-encoders to learn sparse lexical (bag-of-words) representations for text retrieval. For example, Dai and Callan [2020] demonstrate with DeepCT that replacing TF-IDF with contextualized term weights from a transformer-based regression model significantly improves retrieval effectiveness. Subsequent work further combines term reweighting techniques with term expansion to address vocabulary and semantic mismatch issues in lexical representations. Some methods [Mallia et al., 2021, Lin and Ma, 2021, Zhuang and Zucco, 2021b] leverage another model for expansion, which incurs additional costs in both training and inference. As an alternative, Formal et al. [2021a] exploit BERT’s masked language model (MLM) layer to train a single model for both expansion and term weighting.

Compared to dense semantic models, sparse lexical models appear to be more robust to domain shifts [Thakur et al., 2021b]. However, optimizing sparse lexical retrievers must take into account the efficiency of query evaluation using inverted indexes. For example, to achieve better effectiveness, term expansion techniques often make sparse lexical representations denser, sometimes rendering retrieval with inverted indexes much slower [Mackenzie et al., 2023]. Additional performance penalties are incurred when integrating sparse lexical and dense semantic representations into hybrid retrieval systems. Since it is impractical to directly compute dot products between high-dimensional vectors at scale in latency-sensitive retrieval applications (i.e., via brute-force approaches), using lexical representations with inverted indexes remains the only sensible choice, though this comes with the aforementioned limitations.

Motivated by these tradeoffs, we explore an approach for *directly* computing dot products between lexical vectors in a scalable and low-latency manner for retrieval applications. This is achieved by densifying lexical representations, which can be applied to any existing lexical model. The approach consists of two simple steps: (1) representation slicing and (2) sliced representation pooling. This method can be viewed as compressing high-dimensional lexical representations into low-dimensional ones, as depicted in Fig. 5.1. One key feature of our approach is that it does not involve either unsupervised [Johnson et al., 2021] or supervised training [Zhan et al., 2021a, 2022, Chen et al., 2022]. Any collection of lexical representations (from a “base model”) can be converted into dense lexical representations (DLRs), and the nearest neighbor search problem between query and passage vectors can be performed using an operation we call the gated inner product (GIP). Standard dense representations and inner products can be considered special cases of DLRs and GIP, respectively. Thus, our representation and scoring function

Table 5.1: Comparison of single-vector representation learning approaches.

matching type		Sparse	Dense
Lexical	whole words	DeepCT, DeepImpact	DLR (ours)
	wordpiece tokens	uniCOIL, SPLADE	
Semantic		UHD	DPR, ANCE TASB, RocketQA
Hybrid		-	DHR (ours)

comprise a unified framework for *both* lexical and semantic matching.

An advantage of our framework is that end-to-end lexical and hybrid retrieval can be performed *directly* on GPUs with a single index structure in a unified execution environment. In fact, our implementation uses standard vector operations directly within the popular PyTorch open-source neural modeling toolkit. Unlike inverted indexes, retrieval efficiency is not sensitive to the sparsity of representations; thus, we can optimize retrieval effectiveness without any constraints. Building on these features, we propose a new model called Dense Lexical AnD Expansion (DeLADE) and explore a new matching approach, which we refer to as dense lexical matching (shown in Table 5.1).

In addition, we can integrate DLRs with dense semantic representations into what we call dense hybrid representations (DHRs). This can be accomplished in two ways: First, we can independently combine DLRs with any “off-the-shelf” dense semantic representations, such as ANCE [Xiong et al., 2021], TAS-B [Hofstätter et al., 2021], etc. Second, we can *jointly* train DHRs that combine both lexical and semantic components. Both DLRs and DHRs offer advantageous features for real-world applications: (1) end-to-end retrieval can be performed on GPUs using a single index and software framework, instead of, for example, using Lucene for lexical matching and Faiss for semantic matching; (2) vector densification can be applied to meet storage constraints without the need for model retraining.

5.2 Methodology

In this work, we explore an approach to *directly* computing dot products between lexical vectors in a scalable and low-latency manner for retrieval applications. This is accomplished by densifying lexical representations, which can be applied to any existing lexical model. The approach described in this paper is comprised of two simple steps: (1) representation slicing (2)

sliced representation pooling. This method can be viewed as compressing high-dimensional lexical representations into low-dimensional ones, as depicted in Fig. 5.1. One key feature of our approach is that it does not involve either unsupervised [Indyk and Motwani, 1998, Jégou et al., 2011, Ge et al., 2014] or supervised training [Zhan et al., 2021a, 2022, Chen et al., 2022]. Importantly, any collection of lexical representations (from a “base model”) can be converted into dense lexical representations (DLRs) without any sparsification, and the nearest neighbor search problem between query and passage vectors can be performed with an operation we call the gated inner product (GIP). Standard dense representations and inner products can be considered a special case of DLRs and GIP, respectively. Thus, our representation and scoring function comprise a unified framework for *both* lexical and semantic matching.

5.2.1 Dense Lexical Representations (DLRs)

Lexical representations can be considered vectors with $|V|$ dimensions, i.e., $\mathbf{q}_{\text{BoW}} = (q_0, \dots, q_{|V|-1})$ and $\mathbf{p}_{\text{BoW}} = (d_0, \dots, d_{|V|-1})$. Our focus here is to “densify” such representations. We first divide each vector into M slices, each of which is a smaller vector with N dimensions (i.e., $|V| = M \cdot N$). In terms of the standard “slice” notation used by Python:

$$\begin{aligned} S_m^q &= \mathbf{q}_{\text{BoW}}[mN : mN + N] \in \mathbb{R}^N; \\ S_m^d &= \mathbf{p}_{\text{BoW}}[mN : mN + N] \in \mathbb{R}^N, \end{aligned} \quad (5.1)$$

where $m \in \{0, 1, \dots, M-1\}$. Note that the slicing can be performed in different ways; for example, slicing randomly or with a fixed stride: $[m : M(N-1) + m : N]$. For simplicity, we use contiguous slicing as shown in Eq. (5.1) in our presentation. Thus, the inner product between $\mathbf{q}_{\text{BoW}}$ and $\mathbf{p}_{\text{BoW}}$ can be rewritten as the summation of all the dot products of their slices:

$$\langle \mathbf{q}_{\text{BoW}}, \mathbf{p}_{\text{BoW}} \rangle = \sum_{m=0}^{M-1} \langle S_m^q, S_m^d \rangle. \quad (5.2)$$

Intuitively, if a lexical representation is sparse enough, we can assume that for each slice, there is only one non-zero entry. Thus, we can approximate S_m^q (S_m^d) by keeping only the entry with the maximum value in each slice:

$$\begin{aligned} S_m^q &\approx \max(S_m^q) \cdot \hat{\mathbf{u}}(\mathbf{e}_m^q); \\ S_m^d &\approx \max(S_m^d) \cdot \hat{\mathbf{u}}(\mathbf{e}_m^d), \end{aligned} \quad (5.3)$$

where $\hat{\mathbf{u}}(\mathbf{e}_m^q)$ is a unit vector with the only non-zero entry at the entry $\mathbf{e}_m^q = \text{argmax}(S_m^q)$. Thus, the inner product of $\mathbf{q}_{\text{BoW}}$ and $\mathbf{p}_{\text{BoW}}$ lexical vectors in Eq. (5.2) can be approximated as follows:

$$\begin{aligned}\langle \mathbf{q}_{\text{BoW}}, \mathbf{p}_{\text{BoW}} \rangle &\approx \sum_{m=0}^{M-1} \max(S_m^q) \cdot \max(S_m^d) \cdot \langle \hat{\mathbf{u}}(\mathbf{e}_m^q), \hat{\mathbf{u}}(\mathbf{e}_m^d) \rangle \\ &= \sum_{m=0}^{M-1} \max(S_m^q) \cdot \max(S_m^d) \mathbb{1}_{\{\mathbf{e}_m^q = \mathbf{e}_m^d\}}\end{aligned}\quad (5.4)$$

Observing Eq. (5.4), in order to compute the approximate inner product of lexical vectors, each query (passage) can be alternatively represented as two M -dimension dense vectors:

$$\begin{aligned}\mathbf{q}_{\text{DLR}}^{\text{val}} &= (\max(S_0^q), \dots, \max(S_{M-1}^q)) \in \mathbb{R}^M \\ \mathbf{q}_{\text{DLR}}^{\text{idx}} &= (\mathbf{e}_0^q, \dots, \mathbf{e}_{M-1}^q) \in \mathbb{N}^M\end{aligned}\quad (5.5)$$

$$\begin{aligned}\mathbf{p}_{\text{DLR}}^{\text{val}} &= (\max(S_0^d), \dots, \max(S_{M-1}^d)) \in \mathbb{R}^M \\ \mathbf{p}_{\text{DLR}}^{\text{idx}} &= (\mathbf{e}_0^d, \dots, \mathbf{e}_{M-1}^d) \in \mathbb{N}^M,\end{aligned}\quad (5.6)$$

where $\mathbf{q}_{\text{DLR}}^{\text{val}}$ ($\mathbf{p}_{\text{DLR}}^{\text{val}}$) is the dense vector storing the M maximum values from the query (passage) slices, and $\mathbf{q}_{\text{DLR}}^{\text{idx}}$ ($\mathbf{p}_{\text{DLR}}^{\text{idx}}$) is the integer dense vector storing the entries with the maximum value in the corresponding slices.

Formally, a dense lexical representation (DLR) is a pair of vectors comprising a “value” vector and an “index” vector, as indicated in Figure 5.1. For a query, the DLR is $(\mathbf{q}_{\text{DLR}}^{\text{val}}, \mathbf{q}_{\text{DLR}}^{\text{idx}})$ and for each passage, $(\mathbf{p}_{\text{DLR}}^{\text{val}}, \mathbf{p}_{\text{DLR}}^{\text{idx}})$. Intuitively, the “value” vector stores the most important term weight in each slice while the “index” vector stores the position of the corresponding terms in each slice. Using DLRs, Eq. (5.4) can be rewritten as:

$$\langle \mathbf{q}_{\text{BoW}}, \mathbf{p}_{\text{BoW}} \rangle \approx \sum_{m=0}^{M-1} \mathbf{q}_{\text{DLR}}^{\text{val}}[m] \cdot \mathbf{p}_{\text{DLR}}^{\text{val}}[m] \cdot \mathbb{1}_{\{\mathbf{q}_{\text{DLR}}^{\text{idx}}[m] = \mathbf{p}_{\text{DLR}}^{\text{idx}}[m]\}}, \quad (5.7)$$

where $\mathbf{q}_{\text{DLR}}^{\text{val}}[m]$ ($\mathbf{p}_{\text{DLR}}^{\text{val}}[m]$) and $\mathbf{q}_{\text{DLR}}^{\text{idx}}[m]$ ($\mathbf{p}_{\text{DLR}}^{\text{idx}}[m]$) is the m -th entry of the query (passage) DLR. Note that the query (passage) DLR represents approximations of the original lexical query (passage) vector, $\mathbf{q}_{\text{BoW}}$ ($\mathbf{p}_{\text{BoW}}$). Thus, the computation in Eq. (5.7) using DLRs is an approximation of the original inner product between the lexical vectors.

To simplify Eq. (5.7), we define a new operation, *gated inner product* (GIP) as follows:

$$\text{GIP}(\mathbf{q}, \mathbf{d}, \mathbf{g}) \triangleq \sum_{m=0}^{M-1} \mathbf{q}[m] \cdot \mathbf{d}[m] \cdot \mathbf{g}[m], \quad (5.8)$$

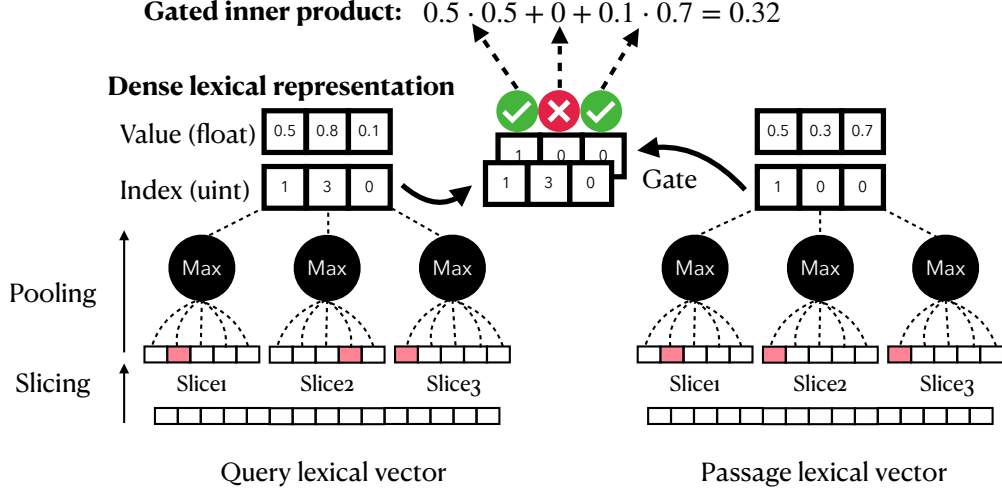


Figure 5.1: Illustration of densified lexical representations (DLRs) and gated inner product (GIP). Both query and passage lexical representations are fixed-width vectors where the number of dimensions is equal to the vocabulary size. Our approach first groups these high-dimensional vectors into M slices, each with N dimensions (e.g., $M = 3$, $N = 5$ here). For each slice, the maximum value is selected. These values from the original vector (Value) and their positions in each slice (Index) are recorded separately. When computing the gated inner product between two vectors, only the dimensions with the same index are considered.

where $\mathbf{g}$ is the gate vector with entries either equal to 0 or 1 (i.e., $\mathbf{g}[m] \in \{0, 1\}$). Thus, we can consider Eq. (5.7) as a GIP operation between query ($\mathbf{q}_{\text{DLR}}^{\text{val}}, \mathbf{q}_{\text{DLR}}^{\text{idx}}$) and passage ($\mathbf{p}_{\text{DLR}}^{\text{val}}, \mathbf{p}_{\text{DLR}}^{\text{idx}}$) DLRs:

$$\langle \mathbf{q}_{\text{BoW}}, \mathbf{p}_{\text{BoW}} \rangle \approx \text{GIP}(\mathbf{q}_{\text{DLR}}^{\text{val}}, \mathbf{p}_{\text{DLR}}^{\text{val}}, \mathbf{g}_{\text{DLR}}), \quad (5.9)$$

where the gate vector $\mathbf{g}_{\text{DLR}}[m] = \mathbb{1}_{\{\mathbf{q}_{\text{DLR}}^{\text{idx}}[m] = \mathbf{p}_{\text{DLR}}^{\text{idx}}[m]\}}$ can be interpreted as the result of lexical matching between two DLRs. This is why lexical matching can still be performed in low-dimensional DLRs and is the key difference between GIP and standard inner product. In summary, DLRs and GIP (representing the original vectors and the inner product, respectively) capture the representation and scoring function in our framework.

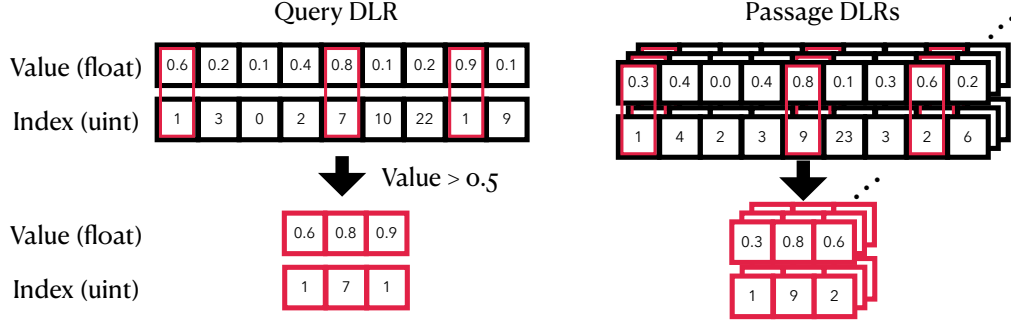


Figure 5.2: Approximate GIP. DLR retrieval can be approximated by only computing the gated inner product from a few dimensions where the query value vector has values above a threshold (e.g., greater than 0.5 here). In two-stage retrieval, we can then precisely rerank the top- K passages using all the dimensions.

5.2.2 Independent Model Fusion

In order to perform retrieval based on the fusion of lexical and semantic representations, we can compute their fusion scores as follows:

$$\text{sim}_{\text{hybrid}} = \phi_{\text{Dot}}(\eta_q^\theta(q), \eta_p^\theta(p)) = \mathbf{q}_{[\text{CLS}]} \cdot \mathbf{p}_{[\text{CLS}]} + \lambda^2 \cdot \mathbf{q}_{\text{BoW}} \cdot \mathbf{p}_{\text{BoW}} \quad (5.10)$$

$$\begin{aligned} &\approx \lambda^2 \cdot \text{GIP}(\mathbf{q}_{[\text{CLS}]}, \mathbf{p}_{[\text{CLS}]}, \mathbf{1}) + \text{GIP}(\mathbf{q}_{\text{DHR}}^{\text{val}}, \mathbf{p}_{\text{DHR}}^{\text{val}}, \mathbf{g}_{\text{DHR}}^{\text{val}}) \\ &= \text{GIP}(\underbrace{\mathbf{q}_{[\text{CLS}]} \oplus \lambda \cdot \mathbf{q}_{\text{DHR}}^{\text{val}}}_{\mathbf{q}_{\text{DHR}}}, \underbrace{\mathbf{p}_{[\text{CLS}]} \oplus \lambda \cdot \mathbf{p}_{\text{DHR}}^{\text{val}}}_{\mathbf{p}_{\text{DHR}}}, \underbrace{\mathbf{1} \oplus \mathbf{g}_{\text{DHR}}^{\text{val}}}_{\mathbf{g}_{\text{DHR}}}), \end{aligned} \quad (5.11)$$

where $\mathbf{1}$ is a vector of all ones with the same dimension as $[\text{CLS}]$ and $\oplus$ is vector concatenation. Note that we rewrite the inner product $\mathbf{q}_{[\text{CLS}]} \cdot \mathbf{p}_{[\text{CLS}]}$ as a special case of their GIP when all the entries in the gate vector are equal to one. Observing Eq. (5.11), the fusion score can be considered a GIP operation between $(\mathbf{q}_{\text{DHR}}^{\text{val}}, \mathbf{q}_{\text{DHR}}^{\text{idx}})$ and $(\mathbf{p}_{\text{DHR}}^{\text{val}}, \mathbf{p}_{\text{DHR}}^{\text{idx}})$:

$$\mathbf{q}_{\text{DHR}}^{\text{val}} = \mathbf{q}_{\text{DHR}}^{\text{val}} \oplus \lambda \cdot \mathbf{q}_{[\text{CLS}]}; \mathbf{q}_{\text{DHR}}^{\text{idx}} = \mathbf{q}_{\text{DHR}}^{\text{idx}} \oplus \mathbf{1}, \quad (5.12)$$

$$\mathbf{p}_{\text{DHR}}^{\text{val}} = \mathbf{p}_{\text{DHR}}^{\text{val}} \oplus \lambda \cdot \mathbf{p}_{[\text{CLS}]}; \mathbf{p}_{\text{DHR}}^{\text{idx}} = \mathbf{p}_{\text{DHR}}^{\text{idx}} \oplus \mathbf{1}. \quad (5.13)$$

We call the pair of vectors, $\mathbf{q}_{\text{DHR}}^{\text{val}}$ and $\mathbf{q}_{\text{DHR}}^{\text{idx}}$ ($\mathbf{p}_{\text{DHR}}^{\text{val}}$ and $\mathbf{p}_{\text{DHR}}^{\text{idx}}$), query (passage) dense hybrid representations (DHRs). Note that DHRs represent a special case of DLRs since they have the same form of representation and scoring function.

5.2.3 Single Model Fusion

We further study whether our framework can benefit from a single model; thus, we propose to *jointly* train lexical and semantic representations within a single model. Inspired by Gao et al. [2021b], our intuition is that such learned representations can better complement each other to perform both lexical and semantic matching.

Given a query q , its relevant passage p^+ and a set of negative passages $\{p_1^-, p_2^-, \dots, p_l^-\}$, we train our model by minimizing negative log likelihood of the positive $\{q, p^+\}$ pair over all the passages:

$$NLL = -\log \frac{e^{\text{sim}_{\text{hybrid}}(q, p^+)}}{e^{\text{sim}_{\text{hybrid}}(q, p^+)} + \sum_{j=1}^l e^{\text{sim}_{\text{hybrid}}(q, p_j^-)}}. \quad (5.14)$$

Following Karpukhin et al. [2020], we also include both negative and positive passages from the other queries in the same batch as the negatives. Note that, we use the exact fusion score $\phi_{\text{dot}}(\eta_q^\theta(q), \eta_p^\theta(p))$ in Eq. (5.10) for training, where the original 30522-dimensional lexical representations are used. However, when performing retrieval, we use the approximate fusion score in Eq. (5.11).

5.2.4 End-to-End Retrieval

While it is possible to perform end-to-end retrieval for each query DLR through GIP computations against all passage DLRs in the corpus, we can identify one weakness. Unlike standard dense representations, GIP requires $4 \cdot M$ operations, which is more than the standard inner product of M -dimensional dense vectors, which only requires $2 \cdot M$ operations. When conducting brute-force search over corpus C , the difference becomes: $4 \cdot M \cdot |C| > 2 \cdot M \cdot |C|$.

To address the above issue, we propose a two-stage retrieval approach inspired by previous work [Khattab and Zaharia, 2020, Yamada et al., 2021]. We first retrieve the top- K candidates (where $K \ll |C|$) using approximate score computations and then rerank the K candidates based on the more accurate GIP computations. In this work, we propose *approximate* GIP for first-stage approximate retrieval. User queries usually contain only a few key terms, which means that when searching the entire corpus, we can perform GIP based on only a few dimensions of DLRs:

$$\text{GIP}(\mathbf{q}_{\text{DLR}}^{\text{val}}, \mathbf{p}_{\text{DLR}}^{\text{val}}, \mathbf{g}_{\text{DLR}}) \approx \sum_{m \in \mathcal{M}} \mathbf{q}_{\text{DLR}}^{\text{val}}[m] \cdot \mathbf{p}_{\text{DLR}}^{\text{val}}[m] \cdot \mathbf{g}_{\text{DLR}}[m], \quad (5.15)$$

where $\mathcal{M} = \{m | \mathbf{q}_{\text{DLR}}^{\text{val}}[m] > \theta\}$ is the set of indices for the GIP computation, and θ is a hyperparameter. This first-stage retrieval relies on the dimensions where $\mathbf{q}_{\text{DLR}}^{\text{val}}[m]$ is above a threshold, as depicted in Figure 5.2.

Note that this approach can also be applied to DHRs, which inherit all their properties from DLRs (and thus can be applied to representations that combine lexical and semantic components). In the subsequent main experiments, we use this retrieve-and-rerank approach with approximate GIP as our first-stage retriever. See Section 5.4.3 for further analyses.

5.2.5 Choice of Lexical Representation Models

Although our approach can be applied to any off-the-shelf model for lexical matching, as described in Section 5.2.2, in practice, many of sparse (lexical) retrieval models still have limitations. For example, uniCOIL [Lin and Ma, 2021] and DeepImpact [Mallia et al., 2021] require another model for passage expansion, which incurs additional costs for training and inference. Thus, in this paper, we choose SPLADE [Formal et al., 2021b] as the basis of our lexical representation model, which addresses the above issue by directly learning term expansions and term weights together. However, Formal et al. [2021b] demonstrate that additional steps are required for tuning a good efficiency–effectiveness tradeoff using a sparsity regularization term [Paria et al., 2020] to enable retrieval with inverted indexes. To address this shortcoming, we propose a variant: playing off the name of SPLADE, which stands for **S**Parse **L**exical **A**nD **E**xpansion, we call this variant the **D**ense **L**exical **A**nD **E**xpansion (DeLADE) model. Instead of regularizing vector sparsity, we increase vector density—the exact opposite. There are two advantages of this design: (1) hyperparameter tuning is not required for model training; (2) the dense vectors are more robust to our densification approach, as we show in Section 5.4.4.

SPLADE. Following SparTerm [Bai et al., 2020], SPLADE generates lexical vectors based on the logits of the BERT pre-trained masked language model (MLM). Consider an input (query or passage) wordpiece sequence $S = (s_1, s_2, \dots, s_n)$ and its corresponding contextualized token embeddings $H = (\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n) \in \mathbb{R}^{n \times 768}$. The logit of each token s_i is computed as follows:

$$\text{logit}_i = \text{transform}(\mathbf{h}_i)^T \cdot E_{\text{MLM}} + b_{\text{MLM}} \in \mathbb{R}^{|V_{\text{BERT}}|}, \quad (5.16)$$

where $\text{transform}(\cdot)$ is a linear layer with ReLU activation and LayerNorm, $E_{\text{MLM}} \in \mathbb{R}^{|V_{\text{BERT}}| \times 768}$ is the BERT embedding table, and b_{MLM} is the bias. V_{BERT} is the vocabulary of BERT wordpiece tokens with size $|V_{\text{BERT}}| = 30522$. SPLADE generates a single-vector embedding $\mathbf{sp}_{\text{BoW}} \in \mathbb{R}^{|V_{\text{BERT}}|}$

by max pooling over sequence token logits.

$$\mathbf{sp}_{\text{BoW}}[v] = \max_{i=1,2,\dots,n} \log(1 + \text{ReLU}(\mathbf{logit}_i[v])), \quad (5.17)$$

where $\mathbf{sp}_{\text{BoW}}[v]$ is the vector value of the index (or vocabulary ID) $v \in [0, |V_{\text{BERT}}|)$ and $\text{ReLU}(\cdot)$ is the activation function. This design naturally promotes vector sparsity since $\log(1 + \text{ReLU}(\cdot))$ becomes zero for all negative input. Together with FLOP regularization loss within a mini-batch $\mathcal{B}$:

$$\mathcal{L}_{\text{FLOP}} = \sum_{v=1}^{|V_{\text{BERT}}|} \left(\frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} \mathbf{sp}_{\text{BoW}}[v] \right)^2, \quad (5.18)$$

the sparsity of $\mathbf{sp}_{\text{BoW}}$ can be further increased. The overall training loss for SPLADE becomes:

$$\mathcal{L} = \text{NLL} + \lambda_q \cdot \mathcal{L}_{\text{FLOP}}^q + \lambda_d \cdot \mathcal{L}_{\text{FLOP}}^d. \quad (5.19)$$

Thus, SPLADE training requires tuning two hyperparameters, λ_q and λ_d , in order to obtain a good effectiveness–efficiency tradeoff. We refer readers to previous work [Paria et al., 2020, Formal et al., 2021b] for more details.

DeLADE. We make a slight revision to Eq. (5.17) by replacing the activation function $\text{ReLU}(\cdot)$ with $\text{softmax}(\cdot)$ to promote vector density.

$$\mathbf{ds}_{\text{BoW}}[v] = \max_{i=1,2,\dots,n} w_i \cdot \text{softmax}(\mathbf{logit}_i)[v], \quad (5.20)$$

where $w_i = \mathbf{h}_i^T W + b$ is the linear transformation of $\mathbf{h}_i$ as the term weight (i.e., capturing term importance) of token s_i ; $\text{softmax}(\mathbf{logit}_i)$ can be interpreted as the contextualized representation of token s_i , as a probability distribution over V_{BERT} . The intuition behind this design is that a sequence is represented as the max pooling of all contextualized representations, while giving more weight to important tokens. Furthermore, in contrast to ReLU , softmax is the activation function used in BERT MLM pre-training, which also ensures that the output vector is dense. Since vector sparsity is not a concern for retrieval latency in our framework, DeLADE can focus on optimizing negative log likelihood loss without tuning additional hyperparameters to balance efficiency considerations.

5.3 Experimental Setups

Datasets. We use the Dev, TREC DL 2019 and 2020 queries in MS MARCO passage ranking dataset for supervised evaluations aforementioned in Section 2.5. As for zero-shot evaluations,

we use BEIR-13 dataset. Model retrieval effectiveness is evaluated in terms of nDCG@10 and R@100, except for TREC-COVID, where we follow Thakur et al. [2021b] and use “capped” Recall@100 instead of “regular” R@100.

DeLADE. In practice, we can jointly train the [CLS] representation with any of the above lexical retrieval models using a single BERT model. However, as discussed in Section 5.2.5, DeepImpact and uniCOIL require an additional model for passage expansion, and thus they are not ideal base models. For SPLADE, FLOP regularization and hyperparameter tuning are required to achieve a good effectiveness–efficiency balance, also making it not an ideal base model. Instead, we choose our proposed DeLADE model as the base lexical model and refer to the single fusion model as DeLADE+[CLS]. The corresponding DHR is called $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(\text{dim})$, where dim denotes the dimensionality of the lexical component, DLR. We project the [CLS] vectors into 128 dimensions with a linear layer.

Densification. We densify high-dimensional representations by first removing some dimensions to make the dimensionality a multiple of 768; then, we further divide each vector into 768 slices. For example, the 30522-dimensional representations are densified into 768-dimensional vectors by (1) discarding the first 570 unused tokens in the BERT vocabulary; (2) dividing the remaining 29952 tokens into 768 slices. Each of the slices contains 39 distinct vocabulary items; that is, $M = 768$ and $N = 39$. In addition, we conduct experiments to densify into 256 and 128 dimensions, where there are 117 and 234 tokens in each slice, respectively. The “value” and “index” dense vectors (see Figure 5.1) are stored as `float16` and `uint8`, respectively.

Training and Inference Details. Our proposed models, DeLADE and $(\text{DeLADE}+[\text{CLS}])$, are implemented using Tevatron [Gao et al., 2023] and trained on a single Tesla V100 GPU with 32 GB memory. We train our models using 6-layer DistilBERT [Sanh et al., 2019] for 6 epochs (around 100k steps) with learning rate $7\text{e-}6$. Each batch includes 24 samples, and for each query, we randomly sample one positive and seven negative passages.² All the negatives are sampled from the MS MARCO “small” triples training set, which is created using BM25. We set the maximum input length for the query and the passage to 32 and 150 (including the special tokens [CLS] and [SEP]), respectively, at both training and inference stages, except for the BEIR dataset, where we set the maximum input length to 512 for both the query and the passage at inference time. For $(\text{DeLADE}+[\text{CLS}])$, we set λ to one during training and inference.

²<https://huggingface.co/distilbert/distilbert-base-uncased>

Training with TCT-ColBERT. To further compare with other state-of-the-art retrievers, we also train our model with knowledge distillation (KD) and hard negative mining (HNM), denoted $(\text{DeLADE}+[\text{CLS}])^+$. Specifically, we use $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(128)$ to retrieve the top-200 passages using the 8M queries in the training set and then retrain $(\text{DeLADE}+[\text{CLS}])$ using ColBERT as the teacher model. Note that our ColBERT model (initialized from DistilBERT) is trained with the soft labels provided by Hofstätter et al. [2020].³ Following Lin et al. [2021c], we use KL divergence as the listwise KD loss, which considers all the in-batch negatives. For each batch, we include 288 triples (i.e., a query with positive and negative examples) by randomly sampling negatives from the top-200 hard negatives. We train $(\text{DeLADE}+[\text{CLS}])^+$ on three Tesla V100 GPUs.

Retrieval Implementation and Settings Operations in our dense representation framework can be implemented by existing packages that support common array operations, which makes our approach easy to implement and to deploy in real-world settings. Specifically, our DLR and DHR retrieval experiments are performed using a custom PyTorch implementation, which means that training and retrieval experiments can be conducted within the same execution environment (on GPUs). Our experiments can be performed without an additional toolkit for nearest neighbor search such as Faiss [Johnson et al., 2021].

As already noted, our DLR and DHR retrieval experiments are conducted in PyTorch directly. For the other dense retrieval models, we use Faiss `FlatIP` GPU indexes as points of comparison. We perform all retrieval experiments using a single NVIDIA RTX A6000 with batch size one. For retrieval using inverted indexes, we use Pyserini (which is built on Lucene) and measure retrieval latency using a single thread on a Linux machine with two 2.1 GHz Intel Xeon Platinum 8160 CPUs and 944G of RAM. Note that in our main experiments, we primarily compare retrieval latency between different types of text representations (i.e., sparse lexical, dense semantic, and DLR/DHR). Thus, we exclude the encoding time of the query text from the neural models for simplicity. Query encoding latency on a GPU is around 10–20 ms per query for the backbones (`bert-base-uncased` or `distilbert-base-uncased`) used by the compared neural models (except for `GTRxxl` [Ni et al., 2022]); thus, this does not have much impact on the latency comparison between neural models. We refer readers to Table 5.8 for detailed online retrieval latency measurements of our models on the CPU and GPU.

Table 5.2: Effectiveness/Efficiency Comparisons of Independent Fusion of DLRs with “Off-the-Shelf” Dense Semantic Representations on MS MARCO (Dev)

Approach	Lexical		Semantic		quality		storage	latency
	Software	Dim	Software	Dim	MRR@10 (diff.)	R@1K (diff.)	(GB)	(ms/q)
(1) Linear combination (BM25, ANCE)	Lucene	30K	Faiss FlatIP	768	0.347	0.969	26	64
(2) SPAR Chen et al. [2022] (lexical conversion to semantic)	n/a	n/a	Faiss FlatIP	2×768	0.344 (−0.9%)	0.971 (+0.2%)	52	81
(3) (BM25+ANCE) _{DHR}	PyTorch	768	PyTorch	768	0.349 (−0.6%)	0.967 (−0.2%)	39	56
	PyTorch	256	PyTorch	768	0.348 (+0.3%)	0.967 (−0.2%)	21	56
	PyTorch	128	PyTorch	768	0.347 (−0.0%)	0.967 (−0.2%)	17	53
(4) Linear combination (uniCOIL, ANCE)	Lucene	30K	Faiss FlatIP	768	0.375	0.976	27	291
(5) SPAR Chen et al. [2022] (lexical conversion to semantic)	n/a	n/a	Faiss FlatIP	2×768	0.369 (−1.6%)	0.981 (+0.5%)	52	81
(6) (uniCOIL+ANCE) _{DHR}	PyTorch	768	PyTorch	768	0.378 (+0.8%)	0.975 (−0.1%)	32	60
	PyTorch	256	PyTorch	768	0.375 (−0.0%)	0.973 (−0.3%)	19	58
	PyTorch	128	PyTorch	768	0.369 (−1.6%)	0.971 (−0.5%)	16	57

Retrieval with DHRs uses our custom PyTorch implementation running on GPUs.

5.4 Experimental Results

5.4.1 Experiments on Dense–Sparse Fusion

Following Chen et al. [2022], we conduct experiments on BM25–ANCE and uniCOIL–ANCE fusion on the MS MARCO Dev set, reported in rows (1)–(3) and rows (4)–(6), respectively, in Table 5.2. The first entry in each main block, rows (1) and (4), represents the linear combination approach used in previous work [Karpukhin et al., 2020, Lin et al., 2021c, Lin and Ma, 2021], which requires two separate indexes and additional post-processing of the ranked lists. For these experiments, we measure latency in Pyserini and report a theoretically optimized system as the maximum latency between Lucene and Faiss retrieval plus 3 ms of post-processing time. We leave aside the engineering challenge of synchronizing CPU and GPU search necessary to achieve this performance under real-world conditions. For SPAR [Chen et al., 2022], rows (2) and (5), we directly report the ranking effectiveness from the paper and measure the retrieval latency of the Faiss FlatIP index in our environment. Note that their approach distills uniCOIL’s lexical representations into semantic representations and then concatenates them to ANCE for dense retrieval. Thus, dimensionality of the representation vectors is reported as 2×768 . For DHRs, rows (3) and (6), we tune λ on MRR@10 using a subset of 100 queries in the training set and set $\theta = 0.3$ for two-stage retrieval.

Overall, all three systems yield similar retrieval effectiveness but our DHRs achieve lower retrieval latency. Note that our model variant with 768-dimensional lexical and semantic vectors

³<https://github.com/sebastian-hofstaetter/neural-ranking-kd>

Table 5.3: Effectiveness/efficiency comparisons. This table compares against a selection of retrieval models trained with comparable baseline strategies. Bold (underline) denotes the best (second best) effectiveness for each row. For the MS MARCO datasets, superscripts denote significant improvements over the labeled model with paired t -test ($p < 0.05$). Note that Bonferroni correction is not applied, so the significance tests may risk overconfidence.

	sparse lexical			dense semantic		multi-vector		DeLADE _{DLR}	(DeLADE+[CLS]) _{DHR}		
	(1) BM25	(2) docT5q	(3) SPLADE	(4) Dense	(5) ANCE	(6) COIL	(7) ColBERT	(8) 768 dim	(9) 128 dim	(a) 256 dim	(b) 768 dim
Efficiency*											
storage (GB)	0.67	0.98	2.6	26	26	60	154	20	5.4	8.6	22
latency (ms/q)	40	64	475	64	64	40*	458*	30	28	31	33
MS MARCO											
Dev MRR@10	0.188	0.277 ¹	0.340 ¹²⁴⁵	0.307 ¹²	0.330 ¹²⁴	0.354 ¹⁻⁵⁸	0.360 [△]	0.345 ¹²⁴⁵	0.351 ¹⁻⁵⁸	0.355 ¹⁻⁵⁸⁹	0.357 ^{1-58-a}
R@1K	0.858	0.947 ¹	0.965 ¹²⁴⁵⁸	0.944 ¹	0.959 ¹²⁴	0.964 ¹²⁴⁵⁸	0.968 [△]	0.953 ¹²⁴	0.962 ¹²⁴⁸	0.965 ¹²⁴⁵⁸⁹	0.967 ^{12458-a}
DL 19 nDCG@10	0.506	0.642 ¹	0.683 ¹⁴	0.631 ¹	0.646 ¹	0.714 ¹²⁴⁵	0.694 [△]	0.691 ¹⁴	0.691 ¹⁴⁵	0.696 ¹⁴⁵	0.693 ¹⁴⁵
DL 20	0.475	0.619 ¹	0.671 ¹	0.648 ¹	0.646 ¹	0.688 ¹²⁵	0.676 [△]	0.668 ¹²	0.683 ¹²	0.686 ¹²	0.688 ¹²
BEIR-13											
	nDCG@10										
TREC-COVID	0.656	<u>0.713</u>	0.661	0.604	0.654	0.668	0.677	0.681	0.695	0.701	0.727
NFCorpus	0.325	0.328	0.322	0.244	0.237	0.331	0.305	<u>0.331</u>	0.319	0.324	0.329
NQ	0.329	0.399	0.469	0.410	0.446	<u>0.519</u>	0.524	0.471	0.476	0.487	0.497
HotpotQA	0.603	0.580	0.640	0.441	0.456	0.713	0.593	0.666	0.664	0.679	<u>0.689</u>
FiQA-2018	0.236	0.291	0.289	0.224	0.295	<u>0.313</u>	0.317	0.294	0.292	0.304	0.312
ArguAna	0.315	0.349	0.445	0.323	0.415	0.295	0.233	0.360	<u>0.423</u>	0.405	0.360
Touche-2020 (v2)	0.367	<u>0.347</u>	0.201	0.185	0.240	0.281	0.202	0.266	0.248	0.259	0.280
Quora	0.789	0.802	0.834	0.750	<u>0.852</u>	0.838	0.854	0.755	0.829	0.830	0.831
DBPedia	0.313	0.331	0.370	0.295	0.281	0.398	0.392	0.376	0.397	<u>0.402</u>	0.404
Scidocs	0.158	<u>0.162</u>	0.149	0.103	0.122	0.155	0.165	0.148	0.146	0.148	0.150
Fever	0.753	0.714	0.740	0.651	0.669	0.840	0.771	0.787	0.750	0.781	<u>0.810</u>
Climate-Fever	0.213	0.201	0.187	0.167	0.198	0.216	0.184	0.202	0.215	<u>0.220</u>	0.229
SciFact	0.665	0.675	0.633	0.479	0.507	0.707	0.671	0.674	0.651	0.670	<u>0.685</u>
Avg. nDCG@10	0.440	0.453	0.458	0.375	0.413	<u>0.483</u>	0.453	0.462	0.470	0.478	0.485

* We report efficiency figures on the MS MARCO passage corpus for comparison.

* These numbers are copied from the original papers, which are measured on multi-GPU systems; thus, they cannot be reproduced in our setup.

△ We do not have these run files; thus, no significance testing is performed against ColBERT.

is faster than SPAR using Faiss GPU. We attribute this improvement to our two-stage retrieval approach (see Section 5.4.3 for more details). It is worth noting that with a negligible effectiveness drop we can further compress the lexical representations to 128 dimensions. Furthermore, we can convert lexical representations into DHRs of any width according to user design requirements *without any model retraining*. This flexibility is one major advantage of our approach.

5.4.2 Experiments on Single Model Fusion

Table 5.3 compares model performance in terms of retrieval effectiveness and efficiency. For efficiency, index size and retrieval latency are measured on the MS MARCO Dev set as the point of reference. The comparison models across the columns are categorized as: (1) sparse lexical

Table 5.4: Component retrieval effectiveness of (DeLADE+[CLS])_{DHR}(768)

Component*	MS MARCO Dev		TREC-COVID		FiQA-2018		SciFact	
	MRR@10	R@1K	NDCG@10	CapR@100	NDCG@10	R@100	NDCG@10	R@100
(1) DeLADE + [CLS]	0.357	0.967	0.727	0.541	0.312	0.591	0.685	0.910
(2) DeLADE	0.294	0.930	0.658	0.501	0.193	0.441	0.684	0.920
(3) [CLS]	0.045	0.494	0.125	0.091	0.037	0.185	0.118	0.454

* Row (1) corresponds to (DeLADE+[CLS])_{DHR}(768) in Table 5.3. Note that rows (2) and (3) are different from the Dense and (DeLADE+[CLS])_{DHR}(768) conditions in Table 5.3, where the separate models are trained independently.

Table 5.5: Ablation of (DeLADE+[CLS])_{DHR}(768) varying the semantic component. Bold denotes the best effectiveness for each column. Superscript denotes significant improvement over the labeled model based on paired t -tests ($p < 0.05$). Note that Bonferroni correction is not applied, so the significance tests may risk overconfidence.

[CLS] dimension	MS MARCO Dev		TREC-COVID		FiQA-2018		SciFact	
	MRR@10	R@1K	NDCG@10	CapR@100	NDCG@10	R@100	NDCG@10	R@100
(1) 0*	0.345	0.953	0.681	0.502	0.294	0.567	0.674	0.920
(2) 128*	0.357 ¹	0.967 ¹	0.727¹	0.541¹	0.312 ¹	0.591 ¹	0.685	0.910
(3) 256	0.358 ¹	0.969 ¹	0.722 ¹	0.530 ¹	0.313 ¹	0.590 ¹	0.683	0.917
(4) 768	0.358 ¹	0.969 ¹	0.717	0.540 ¹	0.318¹	0.601¹	0.680	0.918

* The 0 and 128 variants correspond to DeLADE_{DHR}(768) and (DeLADE+[CLS])_{DHR}(768) in Table 5.3, respectively.

retrievers, including BM25, docT5q [Nogueira and Lin, 2019], and SPLADE [Formal et al., 2021a]; (2) dense semantic retrievers, including our trained baseline dense retriever (denoted Dense with 768-dimensional [CLS] vectors) and ANCE [Xiong et al., 2021]; (3) multi-vector retrievers, including ColBERT [Khattab and Zaharia, 2020] and COIL [Gao et al., 2021a].

For the MS MARCO datasets, we conduct experiments using Pyserini [Lin et al., 2021a] for all models except for ColBERT⁴ and COIL.⁵ For the BEIR datasets, we directly copy numbers from Izacard et al. [2022], except for SPLADE⁶ and COIL.⁵ To determine the statistical significance of our results, we perform paired t -tests ($p < 0.05$) comparing all models except for ColBERT on the MS MARCO datasets. For a fair comparison, Table 5.3 only includes models that use the same baseline training strategy as ours. Thus, we exclude approaches that depend on other models for expansion [Mallia et al., 2021, Lin and Ma, 2021, Zhuang and Zuccon, 2021b], costly training techniques such as knowledge distillation [Qu et al., 2021, Hofstätter et al., 2021,

⁴We copy numbers from Mallia et al. [2021].

⁵We run COIL using the inference code from the authors’ repo at <https://github.com/luyug/COIL>.

⁶We run SPLADE-max using the inference code from the authors’ repo at <https://github.com/naver/splade>.

Wang et al., 2022, Santhanam et al., 2022b, Formal et al., 2021a], or special pre-training [Gao and Callan, 2021, Izacard et al., 2022, Ni et al., 2022], although see Table 5.6 for additional comparisons.

In terms of our proposed models, we report results on three $(\text{DeLADE} + [\text{CLS}])_{\text{DHR}}$ variants by densifying the lexical components into 128, 256, and 768 dimensions. The $(\text{DeLADE} + [\text{CLS}])_{\text{DHR}}$ 128- and 256-dimensional variants can be considered “small vectors”, to compare with single-vector (sparse lexical and dense semantic) retrievers, while the $(\text{DeLADE} + [\text{CLS}])_{\text{DHR}}(768)$ variant can be considered “large vectors”, to compare with multi-vector retrievers. Note that all three variants are derived from the same model. Finally, we report the performance of $\text{DeLADE}_{\text{DLR}}(768)$, i.e., without the incorporation of the $[\text{CLS}]$ vector.

DHRs vs DLRs. We first compare $(\text{DeLADE} + [\text{CLS}])_{\text{DHR}}(768)$ with $\text{DeLADE}_{\text{DLR}}(768)$, columns (8) and (b), to examine the effectiveness of single model fusion. From the results, we see that $(\text{DeLADE} + [\text{CLS}])_{\text{DHR}}(768)$, which incorporates an additional 128-dimensional $[\text{CLS}]$ vector, demonstrates better retrieval effectiveness than $\text{DeLADE}_{\text{DLR}}(768)$ for both in-domain and out-of-domain datasets. Furthermore, $(\text{DeLADE} + [\text{CLS}])_{\text{DHR}}(256)$ outperforms $\text{DeLADE}_{\text{DLR}}(768)$ for all metrics, which suggests that incorporating the $[\text{CLS}]$ vector can mitigate information loss from our densification approach.

To further understand how joint training works for $(\text{DeLADE} + [\text{CLS}])_{\text{DHR}}(768)$, we conduct retrieval using the densified 768-dimensional lexical vectors from DeLADE and 128-dimensional $[\text{CLS}]$ semantic vectors separately. Retrieval effectiveness on the MS MARCO Dev set and three BEIR datasets (TREC-COVID, FiQA-2018, and SciFact) is reported in Table 5.4. We see that each component is far less effective individually than their hybrid fusion. Specifically, the DeLADE and $[\text{CLS}]$ components in $(\text{DeLADE} + [\text{CLS}])_{\text{DHR}}(768)$ are less effective than the independently trained $\text{DeLADE}_{\text{DLR}}(768)$ and Dense models, respectively; see columns (4) and (8) in Table 5.3. This result indicates that joint training yields components that are highly complementary, as designed.

To further explore this complementarity of representation, we vary the dimensionality of the semantic component of $(\text{DeLADE} + [\text{CLS}])_{\text{DHR}}(768)$. We jointly train $\text{DeLADE}_{\text{DLR}}(768)$ with 0, 128, 256, and 768-dimensional $[\text{CLS}]$ vectors separately. Retrieval effectiveness on the MS MARCO Dev set and three BEIR datasets (TREC-COVID, FiQA-2018, and SciFact) is reported in Table 5.5. We see that fusing a small $[\text{CLS}]$ vector (e.g., 128 dimensions) in training improves retrieval effectiveness, indicating that joint training yields complementary lexical and semantic representations within a single model. This is consistent with the component analysis above. However, further increasing the dimensionality of the $[\text{CLS}]$ vector does not appear to yield obvious advantages. This result indicates that a relatively small $[\text{CLS}]$ vector is sufficient to

complement the lexical component.

DHRs vs single-vector models. First, a comparison of single-vector retrievers shows that sparse lexical representations have better storage efficiency than dense semantic representations. For example, docT5q, column (2), with an index less than 1 GB, exhibits better out-of-domain retrieval effectiveness than dense semantic retrievers, columns (4) and (5). Similarly, SPLADE, column (3), outperforms the dense retrievers for both in-domain and out-of-domain conditions with an index of only 2.6 GB. However, SPLADE requires over 7 times the retrieval latency of docT5q and other dense retrievers. As lexical–semantic hybrid representations, DHRs exhibit better retrieval effectiveness with modest retrieval latency and index storage consumption. We see that the 128- and 256-dimensional variants of $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}$, columns (9) and (a), outperform the single-vector retrievers, columns (1) to (5), for both in-domain and out-of-domain datasets in terms of retrieval effectiveness, and furthermore achieves lower query latency. In addition, the two $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}$ variants have modest index sizes; for example, the Dense model, column (4), requires 26 GB to store the MS MARCO passage corpus (using a Faiss FlatIP index) while $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(128)$ only consumes 5.4 GB.

DHRs vs multi-vector models. We compare retrieval models with large vectors, multi-vector and $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(768)$ retrieval models, shown in columns (b), (6), and (7). Although their in-domain retrieval effectiveness does not appear to be very different, ColBERT falls behind COIL and $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(768)$ in out-of-domain evaluation. This result suggests that lexical matching remains a key component for generalization. Compared to COIL, our model, $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(768)$, exhibits comparable retrieval effectiveness but has a much smaller index (60 GB vs 22 GB).

It is worth mentioning that COIL has other variants with smaller indexes; for example, the configuration with 8-dimensional tokens plus 128-dimensional [CLS] embeddings consumes 14 GB and yields 0.347 (0.956) MRR@10 (R@1K) on the MS MARCO Dev queries.⁷ This variant is still slightly less effective than $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(256)$ and $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(128)$ with index sizes of 8.6 GB and 5.4 GB, respectively. Furthermore, all our variants are derived from the same model without retraining. This comparison demonstrates the advantages of single model fusion under our proposed dense representation framework.

Summary. We observe that dense retrievers generally perform well in domain, while sparse retrievers appear to yield stronger generalization capabilities. Our results demonstrate that DHRs

⁷We refer readers to Gao et al. [2021a] for more details.

Table 5.6: Effectiveness/efficiency comparisons with existing state-of-the-art models using advanced training techniques. Bold (underline) denotes the best (second best) effectiveness for each row.

	sparse lexical		dense semantic					multi-vector		(DeLADE+[CLS]) ⁺ _{DHR}		
	(c) SPLADEv2	(d) TAS-B	(e) Contriever	(f) RocketQAv2	(g) GPL	(h) GTR _{su}	(i) GTR _{xxl}	(j) ColBERTv2	(k) 128 dim	(l) 256 dim	(m) 768 dim	
model size	66M	66M	110M	110M	66M×13*	1.24B	4.8B	110M			66M	
IR pre-training	✗	✗	✓	✗	✗	✓	✓	✗			✗	
KD	✓	✓	✗	✓	✓	✓	✓	✓			✓	
HNМ	✓	✗	✓	✓	✓	✓	✓	✓			✓	
batch size > 1K	✗	✗	✓	✓	✗	✓	✓	✗			✗	
Efficiency*												
storage (GBs)	5.0	26	26	26	26	26	26	29	5.4	8.6	22	
latency (ms/q)	2864	64	64	64	64	64	64	260	28	31	33	
MS MARCO												
Dev	MRR@10	0.368	0.347	0.341	0.381	-	0.385	<u>0.388</u>	0.397	0.366	0.370	0.371
	R@1K	<u>0.979</u>	0.978	0.979	0.981	-	<u>0.989</u>	0.990	0.984	0.973	0.975	0.977
DL 19	nDCG@10	0.729	<u>0.717</u>	0.678	-	-	-	-	0.703	0.711	0.708	
DL 20		0.710	0.686	0.661	-	-	-	-	0.684	0.696	<u>0.700</u>	
BEIR-13												
		nDCG@10										
TREC-COVID	0.710	0.481	0.596	0.675	0.700	0.584	0.501	0.738	0.686	0.702	<u>0.735</u>	
NFCorpus	0.334	0.319	0.328	0.293	0.345	<u>0.343</u>	0.342	0.338	0.327	0.332	0.337	
NQ	0.521	0.463	0.498	0.505	0.483	0.559	0.568	<u>0.562</u>	0.497	0.512	0.523	
HotpotQA	0.684	0.584	0.638	0.533	0.582	0.591	0.599	0.667	0.659	<u>0.673</u>	0.684	
FiQA-2018	0.336	0.300	0.329	0.302	0.344	<u>0.444</u>	0.467	0.356	0.320	0.326	0.335	
ArguAna	0.479	0.429	0.446	0.451	0.557	0.531	<u>0.540</u>	0.463	0.475	0.458	0.436	
Tóuche-2020 (v2)	0.272	0.162	0.230	0.247	0.255	0.230	0.256	<u>0.263</u>	0.226	0.237	0.254	
Quora	0.838	0.835	0.865	0.749	0.836	<u>0.890</u>	0.892	0.852	0.846	0.848	0.849	
DBPedia	<u>0.435</u>	0.384	0.413	0.356	0.384	0.396	0.408	0.446	0.402	0.409	0.413	
Scidocs	0.158	0.149	<u>0.165</u>	0.131	0.169	0.159	0.161	0.154	0.156	0.158	0.158	
Fever	0.786	0.700	0.758	0.676	0.759	0.717	0.740	0.785	0.764	<u>0.794</u>	0.815	
Climate-Fever	0.235	0.228	0.237	0.180	0.235	0.270	<u>0.267</u>	0.176	0.222	0.232	0.239	
SciFact	0.693	0.643	0.677	0.568	0.674	0.635	0.662	0.693	0.674	<u>0.694</u>	0.699	
Avg.nDCG@10	0.499	0.437	0.475	0.436	0.486	0.488	0.493	0.499	0.481	0.490	<u>0.498</u>	

* Wang et al. [2022] trained one expert model for each BEIR dataset using pseudo-relevant labels from a cross-encoder model.

* We report efficiency figures on the MS MARCO passage corpus for comparison. As detailed in Mackenzie et al. [2023], SPLADEv2 is much slower than the other models with Lucene; even with PISA, a much faster query evaluation implementation, latency is still 220 ms/q. As for ColBERTv2, the number is copied from Vanilla ColBERTv2 (p=4, c=2¹⁶) measured by Santhanam et al. [2022a].

inherit advantages of both lexical and semantic matching. Specifically, our (DeLADE+[CLS])_{DHR} model achieves competitive retrieval effectiveness in both evaluation settings with low retrieval latency and modest index storage consumption. This advantageous effectiveness–efficiency tradeoff makes the design of single model fusion with DHRs attractive. It is also worth noting that DeLADE_{DLR}(768) outperforms SPLADE slightly,⁸ especially in term of nDCG. This result shows that DeLADE is a good alternative under our framework compared to SPLADE using inverted indexes in the scenario where lower retrieval latency is more important than index size.

⁸Our reproduced numbers are slightly better than Formal et al. [2021a] except for Tóuche-2020, where we use v2 instead of v1.

DHRs vs more advanced retrieval models. Finally, we compare (DeLADE+[CLS])_{DHR}⁺, which uses advanced training strategies, to the effectiveness and efficiency of existing state-of-the-art retrievers in Table 5.6. For sparse lexical retrieval models, we report results from SPLADEv2 [Formal et al., 2021a], which uses two rounds (i.e., training with BM25 and hard negatives) of knowledge distillation from a cross-encoder teacher.

For dense semantic retrieval models, we include three representative models for comparison: (1) TAS-B [Hofstätter et al., 2021] distills knowledge from multiple teachers using a topic-aware negative sampling strategy. (2) RocketQAv2 [Ren et al., 2021] leverages a student–teacher joint training approach to make the dense retriever better mimic a cross-encoder teacher. (3) Contriever [Izacard et al., 2022] leverages pre-training by combining advanced contrastive learning techniques with an Inverse Cloze Task (ICT) variant. We also include dense retrievers trained with much larger backbone models (i.e., more model parameters): (1) GPL [Wang et al., 2022] trains an expert model for each target dataset in BEIR. (2) GTR [Ni et al., 2022] trains even larger encoder models; the authors’ T5-3B and T5-11B models correspond to GTR_{XL} and GTR_{XXL}, which also combine pre-training, knowledge distillation (KD), and hard negative mining (HNM).⁹ Finally, for multi-vector retrieval models, we report ColBERTv2 [Santhanam et al., 2022b], which also combines KD and HNM, and at the same time compresses the multi-vectors into a smaller index.

We point out that it is not easy to fairly compare models with more advanced and costly training techniques since, as shown in Table 5.6, there are many substantive differences that cannot be captured by general descriptive labels. Even considering a general method such as knowledge distillation, there are many different implementations. For example, we use a lightweight ColBERT teacher, while GPL, ColBERTv2, and SPLADEv2 use a more expensive cross-encoder teacher. While end-to-end results are comparable since the evaluations use the same test collections, it is difficult to attribute effectiveness differences to specific components.

Nevertheless, it is possible to draw some conclusions from these experiments. We first observe that dense semantic vectors from smaller models, in columns (d)–(f), do not appear to perform well in both in-domain and out-of-domain evaluations. For example, Contriever performs well on BEIR but lags behind most models on in-domain evaluation and the reverse trend can be observed for RocketQAv2. To improve the generalization capability of dense retrieval models, existing work either leverages multiple expert models for multi-domain datasets as GPL, column (g), or larger pre-trained models as GTR, columns (h)–(i).

On the other hand, SPLADEv2 and ColBERTv2, columns (c) and (j), leverage representations with more expressive capacity than dense semantic vectors and appear to show equally

⁹GTR is fine-tuned on MS MARCO training queries with hard negatives denoised by a cross-encoder.

good generalization capability without adding model parameters. However, these models sacrifice retrieval efficiency to gain this generalization capability. For example, both SPLADEv2 and ColBERTv2 are slower than dense semantic models. GTR also sacrifices query encoding latency, which is excluded in our latency measurement.¹⁰

In contrast, $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}^+(768)$ not only yields competitive zero-shot retrieval effectiveness on BEIR compared to GTR_{XXL} , ColBERTv2, and SPLADEv2, but also maintains low retrieval latency. In addition, our $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}^+(256)$, column (l), still yields better overall performance (i.e., effectiveness and efficiency) compared to the single-vector dense retrievers, TAS-B, GPL, and Contriever. We notice that GTR performs particularly well in some QA datasets (e.g., NQ and FiQA-2018), likely because GTR includes more training data such as QA pairs mined from the web and the NaturalQuestions dataset. We also note that $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}$ cannot compete with RocketQAv2, GTR, and ColBERTv2 on in-domain evaluation since the first two exploit additional QA training pairs and all of them use more expensive cross-encoder teachers, which is orthogonal to our work. It is likely that we can further boost $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}$ effectiveness by leveraging more data and more expensive teachers; however, we leave these explorations for future work.

5.4.3 Analysis of End-to-End Retrieval

To illustrate how well our proposed approximate GIP operation compares to the more expensive exact GIP operation between DLRs (or DHRs), we conduct end-to-end retrieval experiments using $\text{DeLADE}_{\text{DLR}}(768)$ and $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(768)$ on the MS MARCO Dev queries.

Figure 5.3 illustrates recall at different cutoffs using approximate GIP with various settings of the parameter θ . Exact GIP, which is equivalent to $\theta = 0$, is shown as the black dashed line and represents the upper bound; as θ increases, recall drops, as expected. However, the results show that the top-10000 candidates retrieved using approximate GIP include more relevant passages (i.e., has higher recall) than GIP at cutoff 1000. That is, the effectiveness drop from approximate (first-stage) retrieval can be remedied by reranking the top-10000 candidates with GIP. In addition, we note that $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(768)$ suffers a smaller recall drop at the approximate retrieval stage. This suggests that [CLS] vectors can help capture relevance and compensate for lower recall from the lexical component with larger values of θ . To be clear, θ is also applied to the [CLS] vector to guide the approximate computations in $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}$. In addition, we also perform retrieval by computing the standard inner product (IP) between two DLRs’ or DHRs’ value vectors (without considering their index vectors) for comparison. We observe that

¹⁰GTR_{XL} and GTR_{XXL} query encoding consume 96 and 349 ms, respectively, as opposed to 10 ms for TAS-B, reported by Ni et al. [2022]

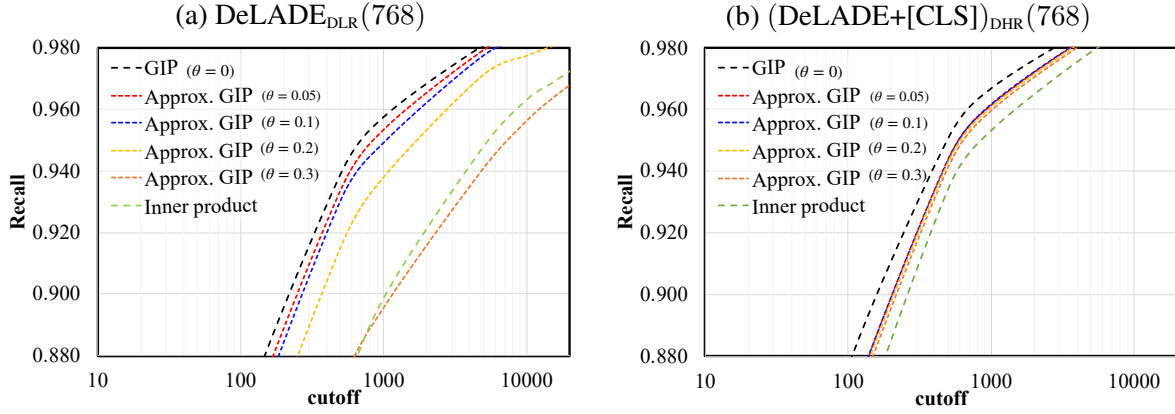


Figure 5.3: Recall comparison of approximate retrieval approaches at different cutoffs. Approx. GIP refers to approximate gated inner product with threshold θ . Inner product refers to standard inner product between query and passage value vectors without involving the index vectors.

IP, surprisingly, also retrieves more relevant passages at cutoff 10000 than GIP at cutoff 1000, especially for DHRs. This indicates that standard inner product between two DLRs’ (or DHRs’) value vectors can be an alternative to approximate GIP. We show below that using IP as first-stage retrieval can benefit DLRs (or DHRs) in end-to-end retrieval on the CPU.

We further examine effectiveness–efficiency tradeoffs after reranking the top-10000 candidates with exact GIP. The red lines in Figures 5.4(a) and 5.4(b) plot the $\text{DeLADE}_{\text{DLR}}(768)$ performance tradeoff curve (ranking effectiveness vs retrieval latency) of approximate GIP and reranking with different values of θ , while the blue dashed line depicts the performance curve of approximate GIP without reranking for comparison. Retrieval effectiveness with exact GIP (the black triangle) can be considered the upper bound. From the blue dashed lines, we observe that approximate GIP substantially reduces retrieval latency but sacrifices effectiveness. However, reranking the top-10000 candidates with GIP mostly recovers the effectiveness loss (except for $\theta = 0.3$) and requires only an additional 5–10 ms.

By incorporating the [CLS] vector, the performance tradeoff curves appear to be even better, as shown in Figures 5.4(c) and 5.4(d), which are organized in the same manner as Figures 5.4(a) and 5.4(b). For $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(768)$, we see no obvious retrieval effectiveness drop, even at $\theta = 0.3$. This result is consistent with our observation in Figure 5.3 that $\text{DeLADE}_{\text{DLR}}(768)$ and $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(768)$ retrieve enough relevant passages in the top-10000 candidates using approximate GIP with $\theta \leq 0.3$. Finally, we observe that IP only shows a minor R@1K degradation after GIP reranking, indicating that the standard inner product can provide an alternative to approximate GIP. This means that end-to-end retrieval with DLRs or DHRs can also be

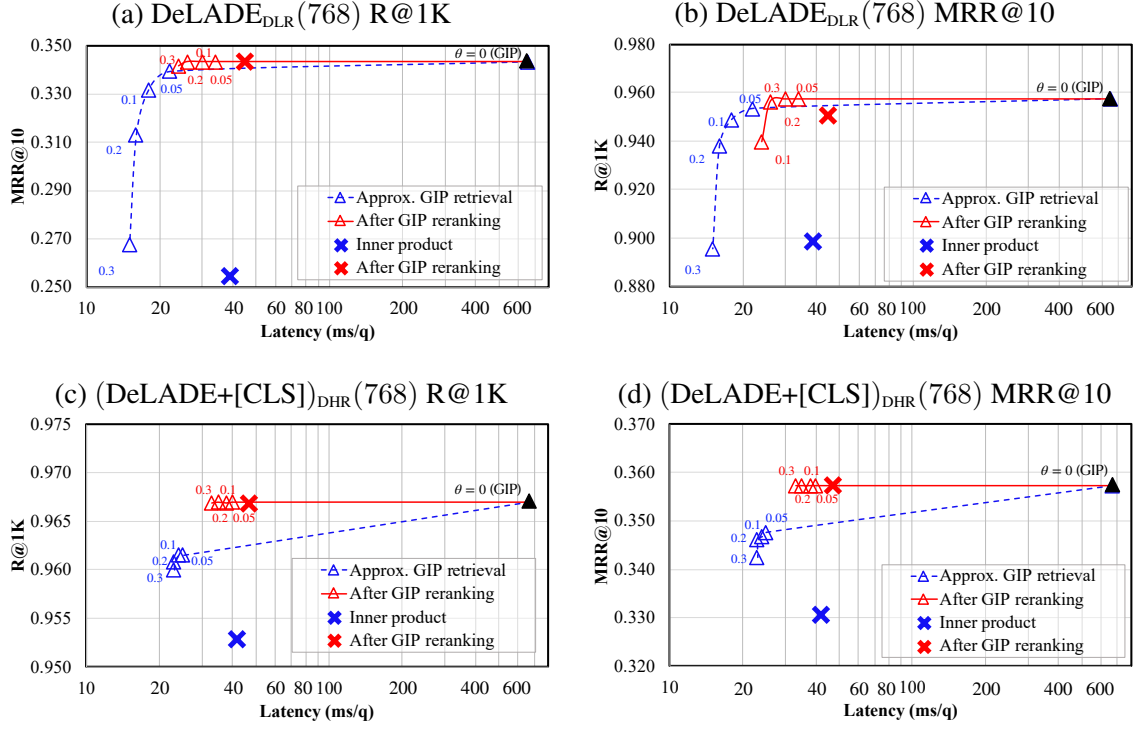


Figure 5.4: Two-stage retrieval performance comparisons. We compare retrieval latency and effectiveness between approximate gated inner product (Approx. GIP) and standard inner product retrieval (between query and passage value vectors without involving the index vectors) and their performance after GIP reranking the top-10000 retrieved candidates. The labeled numbers denote the threshold θ for Approx. GIP retrieval. Note that approx. GIP with $\theta = 0$ is equal to GIP retrieval without any approximation.

implemented with first-stage IP followed by GIP reranking.

In our final analysis, we demonstrate the interpretability of DLRs, which contrasts with dense semantic representations, where it is often difficult to understand why certain query–passage pairs obtain high scores. Since the “index” vector stores the position of the most important term in each slice, a DLR can be “reverted” back into a bag of words with term weights. We showcase how θ impacts approximate GIP in Table 5.7, where we reconstruct expanded query terms from DeLADE_{DLR}(768), shown in the top portion of the table. In addition, we reconstruct the passage judged as relevant to the query in the bottom portion of the table. For simplicity, we do not show term weights.

Table 5.7: Effects of θ on Approximate GIP with the Reconstructed Expanded Query from DeLADE_{DLR}(768)

Original query: where was the bauhaus built		
θ	rank	Reconstructed (expanded) query terms
0.00	2	location, built, ##uh, ##aus, ba, was, house, build, building, site, were, school, home, church, the, store, originally, place, construction, studio, founded, headquarters, structure, later, city, is, origin, be, theater, college, first, hotel, villa, manufacture ... (omit)
0.05	3	location, built, ##uh, ##aus, ba, was, house, build, building, site, were, school
0.10	6	location, built, ##uh, ##aus, ba, was, house, build, building, site
0.20	16	location, built, ##uh, ##aus, ba, was, house, build
0.30	19	location, built, ##uh, ##aus, ba, was, house
Original passage (relevant)		
So the built output of Bauhaus architecture in these years is the output of Gropius: the Sommerfeld house in Berlin, the Otte house in Berlin, the Auerbach house in Jena, and the competition design for the Chicago Tribune Tower, which brought the school much attention. taatliches Bauhaus, commonly known simply as Bauhaus, was an art school in Germany that combined crafts and the fine arts, and was famous for the approach to design that it publicised and taught. It operated from 1919 to 1933.		
Reconstructed relevant (expanded) passage terms		
ba, ##uh, ##aus, ta, output, house, was, berlin, year, 1933, school, help, ##liche, germany, tribune, architecture, jena, chicago, ##at, design, 1919, ##ius, tower, competition, ot, somme, known, ##op, info, art, ##bach attention, an, ##s, ##rf, location, commonly, ##a, simply, is, for, au, combined, ##te, built, operated, ##er, as, famous, ##ised, brought, and, approach, period, public, build, in, fine, building, be, reason, crafts, taught, bring, greatest, much, were, that, combine, skyscraper, architectural, fact, most, production, largest, are, origin, definition, arts, term, so, during, german, name, institution, called, abbreviation, it, type, ##t, which, highest, widely, history, of, important, work, operate, war, the, include, state, difference, organization, its, designed, ##st, company, great, century, 1920, recent, purpose, sculpture, acronym, meaning, studio, produced, last, lot, notable, run, literally, concept, 1934, college, place, created, record, historical, 1918, clock, major, see, time, decorative, produce, say, being, biggest, villa, fifty, many, founded, structure, culture ... (omit)		

The matching terms between the queries and the original (expanded) passage are colored with blue (red). The “rank” column denotes the rank of the relevant passage under different θ settings. The reconstructed query and passage terms are ordered by their term weights in descending order.

We observe that when $\theta = 0$, there are more than 10 matching terms (colored terms) between the query and the passage. As θ increases, terms with lower weights are filtered out; see Eq. (5.15). Thus, the relevance score between the query and the passage decreases. For example, from $\theta = 0$ to 0.05, terms with weights lower than 0.05 are removed for retrieval and the rank of the passage degrades slightly. At $\theta = 0.3$, four important terms (e.g., ‘build’, ‘building’, ‘were’,

Table 5.8: End-to-end retrieval latency for $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}(768)$ on MS MARCO (Dev). Retrieval latency is measured in terms of ms/q with a single thread and batch size 1. With the exception of row (1), for each query, we retrieve the top-10000 passages at the first stage and then rerank the candidates using the second stage. Approx. GIP refers to approximate gated inner product with threshold $\theta = 0.3$. Inner product refers to standard inner product between query and passage value vectors without involving the index vectors.

Two-stage retrieval		CPU latency (ms/q)				GPU latency (ms/q)				Quality	
1st	2nd	Q Enc.	1st	2nd	Total	Q Enc.	1st	2nd	Total	MRR@10	R@1K
(1) GIP	-	164	72560	0	72724	12	670	0	682	0.357	0.967
(2) Approx. GIP	GIP	164	9013	102	9279	12	23	10	45	0.357	0.967
(3) Inner Product	GIP	164	3428	102	3694	12	37	10	59	0.357	0.967
(4) Inner Product (w/ PQ128)	GIP	164	562	102	828	12	-*	10	-*	0.357	0.965
(5) Inner Product (w/ PQ64)	GIP	164	284	102	650	12	-*	10	-*	0.357	0.961

* Faiss.IndexPQ does not support GPU search.

‘school’) are removed, and the passage is retrieved much lower in the ranked list (at rank to 19).¹¹ This example illustrates that tuning θ determines how many query terms (dimensions) are used for approximate GIP.

Next, we measure the end-to-end retrieval latency of $(\text{DeLADE}+[\text{CLS}])_{\text{DHR}}$, including both query encoding (including vector densification) and retrieval components, using the same GPU and CPU environments described in Section 5.3. We report the query latency averaged over the 6980 MS MARCO Dev queries in Table 5.8. With the exception of row (1), for each query, we retrieve the top-10000 passages at the first stage and rerank the candidates using the second stage.

On the GPU, query encoding consumes 12 ms per query. With approximate GIP retrieval and GIP reranking, row (2), our lowest end-to-end retrieval latency on the GPU is 45 ms per query. With inner product retrieval and GIP reranking, query latency rises slightly to 59 ms per query, row (3). End-to-end retrieval latency on the CPU can be reduced to less than one second per query by performing inner product first-stage retrieval on the quantized query and passage value vectors, shown in rows (4) and (5). Although approximate GIP can improve the latency of first-stage retrieval over $80\times$ compared to GIP on the CPU, row (2) vs row (1), approximate GIP is still over $2\times$ slower than inner product, row (2) vs row (3). This stands in contrast to the GPU, where approximate GIP is *faster* than inner product. We note that approximate GIP requires selection of certain dimensions, i.e., $\mathcal{M}$ in Eq. (5.15), from the passage value and index

¹¹We notice some “wacky” terms from the reconstructed passage with high term weights, originally observed by Mackenzie et al. [2023].

Table 5.9: Comparison of Vector Densification Settings for the MS MARCO Passage Corpus

Model	vocabulary size	discarded vocabulary	value vector type	index vector type
BM25	2,660,824	472	float16	uint16
DeepImpact	3,514,102	502	float16	uint16
uniCOIL/SPLADE/DeLADE	30,522	570	float16	uint8
Dense [CLS]	-	-	float16	*

* Note that we do not have to store the index vector for [CLS] since it is a vector of all ones.

vectors across the entire corpus, and this operation appears to be the latency bottleneck in our implementation using PyTorch on the CPU with a single thread. Thus, on the CPU, using the inner product between query and passage vectors is a better choice for first-stage retrieval, which can be further accelerated with approximate nearest neighbor (ANN) search algorithms.

5.4.4 Discussions

How well do DLRs approximate any existing high-dimensional lexical representations?

To evaluate our approach to densifying lexical representations, we conduct experiments on four lexical matching models: two models based on whole word matching, with large vocabulary sizes (millions of distinct terms), and three models that operate on the wordpiece vocabulary of BERT (30522 distinct tokens), including our proposed DeLADE model.

The whole word matching approaches in more detail:

1. BM25: We can characterize this classic retrieval model as generating heuristically assigned term weights.
2. DeepImpact [Mallia et al., 2021]: Term expansion is first applied to each passage in the collection using doc2query-T5 [Nogueira and Lin, 2019]. Then, the encoder (a two-layer MLP with ReLU activation) projects token embeddings from the final layer of BERT into contextualized term weights for each input token in the query and expanded passage.

For BM25 and DeepImpact, we output term weights for each query and passage using Pyserini [Lin et al., 2021a] and then randomly assign each unique term to a unique vocabulary ID to form a high-dimensional sparse vector.

We describe the wordpiece token matching approaches in more detail:

1. uniCOIL [Lin and Ma, 2021]¹²: This model is similar to DeepImpact, but one main difference is that uniCOIL performs lexical matching on BERT wordpiece tokens during both training and retrieval.
2. SPLADE [Formal et al., 2021a]¹³: To be precise, we refer to the SPLADE-max model, which uses BERT pre-trained MLM to project query (or passage) tokens into a $|V_{\text{BERT}}|$ -dimensional sparse lexical representation, where $|V_{\text{BERT}}| = 30522$ is the vocabulary size of BERT wordpiece tokens.
3. DeLADE: Our proposed variant of SPLADE encodes queries and passages into $|V_{\text{BERT}}|$ -dimensional dense lexical representations. We call this variant the Dense Lexical AnD Expansion (DeLADE) model, playing off the name of SPLADE, which stands for Sparse Lexical AnD Expansion.

We refer to the DLRs from a base model as $\text{model}_{\text{DLR}}(\text{dim})$, where dim denotes the dimensionality of the DLR. For example, $\text{uniCOIL}_{\text{DLR}}(768)$ refers to DLRs of 768 dimensions using uniCOIL as the base model. Note that since we randomly assign a vocabulary ID to each word for BM25 and DeepImpact, the discarded words are also randomly chosen. In addition, `uint8` is sufficient to represent the index vectors for the lexical models that rely on wordpiece token matching, while `uint16` is required for BM25 and DeepImpact. The detailed settings for densifying the vectors from different models are shown in Table 5.9. In addition, we also list the standard dense [CLS] vector storage setting under our framework for comparison.

Table 5.10 shows the results of densifying different lexical representations. In these experiments, we use stride slicing, but examine the impact of different slicing strategies below. The first row in each block reports the retrieval effectiveness of the original lexical representations using inverted indexes, which can be considered an upper bound. We also report the effectiveness difference (shown as a percentage) between each method and the upper bound using inverted indexes. Note that since DeLADE is trained without any sparsity constraints, the output query and passage representations are quite dense and hence impractical for retrieval using inverted indexes. For this condition, we use `faiss.FlatIP` brute-force search instead. Index size is provided only as a reference, and we omit query latency since it is not comparable to retrieval with inverted indexes. We also report the number of tokens per passage (i.e., vector indices with non-zero weights) for each vector densification condition.

Our method is able to densify high-dimensional lexical vectors into 768-dimensional vectors with only a small retrieval effectiveness drop. As the vectors are further densified into smaller

¹²<https://huggingface.co/castorini/unicoil-msmarco-passage>

¹³https://github.com/naver/splade/tree/main/weights/splade_max

Table 5.10: Effectiveness/efficiency comparisons of DLRs with different base models on MS MARCO (Dev).

	Base model	Method	Dim	# tokens/doc	Quality		Storage	Latency
				number (diff.)	MRR@10 (diff.)	R@1K (diff.)	GB	ms/q
whole word	BM25	Inv. index	2.6M	30.11	0.188	0.858	0.7	40
		SPAR [Chen et al., 2022]	768	-	0.173 (-8.0%)	0.831 (-3.1%)	26.0	64
		DLR	768	29.18 (-3.1%)	0.180 (-4.3%)	0.845 (-1.5%)	26.0	26
			256	28.41 (-5.7%)	0.177 (-5.9%)	0.834 (-2.8%)	8.6	23
			128	26.62 (-11.6%)	0.169 (-10.1%)	0.816 (-4.9%)	4.3	22
	DeepImpact	Inv. index	3.5M	71.61	0.327	0.948	1.4	285
		DLR	768	65.28 (-8.9%)	0.324 (-0.9%)	0.942 (-0.6%)	26.0	26
			256	59.90 (-16.4%)	0.316 (-3.4%)	0.933 (-1.6%)	8.6	24
			128	52.74 (-26.4%)	0.304 (-7.0%)	0.923 (-2.6%)	4.3	22
	wordpiece token	uniCOIL	Inv. index	30K	67.96	0.351	0.958	1.3
SPAR [Chen et al., 2022]			768	-	0.341 (-2.8%)	0.970 (+1.2%)	26.0	64
DLR			768	64.15 (-5.6%)	0.349 (-0.6%)	0.957 (-0.1%)	20.0	25
			256	58.62 (-13.7%)	0.344 (-2.0%)	0.952 (-0.6%)	6.4	22
			128	52.48 (-22.8%)	0.335 (-4.6%)	0.944 (-1.5%)	3.3	22
SPLADE		Inv. index	30K	91.50	0.340	0.965	2.6	475
		DLR	768	86.33 (-5.7%)	0.336 (-1.2%)	0.963 (-0.2%)	20.0	28
			256	76.45 (-16.5%)	0.326 (-4.2%)	0.959 (-0.6%)	6.4	25
			128	64.35 (-29.7%)	0.318 (-6.5%)	0.951 (-1.5%)	3.3	24
DeLADE		FlatIP*	30K	30522	0.347	0.957	1033.3	-
	DLR	768	768 (-97.5%)	0.345 (-0.6%)	0.953 (-0.4%)	20.0	25	
		256	256 (-99.2%)	0.341 (-1.7%)	0.951 (-0.6%)	6.4	22	
		128	128 (-99.6%)	0.335 (-3.5%)	0.945 (-1.3%)	3.3	21	

* Index size with `faiss.FlatIP` is provided only as a reference; query latency is not comparable to retrieval with inverted indexes and hence omitted.

dimensions, retrieval effectiveness drops more, which can be explained by information loss since the number of tokens are reduced through max pooling in each slice; i.e., collision between tokens appearing in the same slice. Compared to models that use whole word matching, BM25 and DeepImpact, wordpiece matching models appear to be more robust to vector densification. For example, with 128-dimensional DLRs, uniCOIL sees only 4.6% and 1.5% degradation in MRR@10 and R@1K, respectively, while BM25 sees 10% and 4.9% degradation in MRR@10 and R@1K, respectively. In contrast to whole word matching models, wordpiece matching models represent many terms with multiple wordpiece tokens; thus, effectiveness is less sensitive to vector densification since there is more redundancy in the representation. In addition, BM25 is the only model that does not benefit from passage expansion; thus, reducing the token space causes greater effectiveness loss.

Among wordpiece matching models, uniCOIL sees only modest retrieval effectiveness degradation for 256 and 128 dimensions while SPLADE sees larger effectiveness drops. This is likely

Table 5.11: Comparison of Different DLR Slicing Strategies on MS MARCO (Dev).

Strategy	uniCOIL _{DLR} (768)			SPLADE _{DLR} (768)			DeLADE _{DLR} (768)		
	# tokens/doc	MRR@10	R@1K	# tokens/doc	MRR@10	R@1K	# tokens/doc	MRR@10	R@1K
Contiguous	39.27	0.334	0.947	68.49	0.326	0.952	768	0.332	0.949
Stride	64.15	0.349	0.957	86.33	0.336	0.963	768	0.345	0.953
Random	64.32	0.349	0.957	87.91	0.336	0.963	768	0.345	0.953

because SPLADE represents each passage with more wordpiece tokens than uniCOIL does; thus, there are more collisions as the vectors are densified into smaller dimensions. In contrast to SPLADE, although there are more collisions from vector densification with DeLADE_{DLR}, our model sees less retrieval effectiveness degradation. We attribute this robustness to collisions to DeLADE’s full expansion over the entire wordpiece vocabulary space without any sparsity regularization; that is, there appears to be more “redundancy” in the representations. This result indicates that DeLADE is a better alternative to SPLADE for dense lexical matching.

Finally, a comparison of DLR and SPAR [Chen et al., 2022] shows the advantages of our approach. Without any training, our 256-dimensional DLRs applied to BM25 and uniCOIL are able to compete with SPAR. Furthermore, our approach consumes less space and exhibits lower retrieval latency. Note that the index vectors for the whole word matching models (BM25 and DeepImpact) are stored in `unit16` due to larger vocabulary sizes; thus, they consume more storage compared to the wordpiece matching models.

In addition, we observe that retrieval latency using inverted indexes is sensitive to the average number of tokens per passage (i.e., vector sparsity). For example, a 35% increase from 68 to 92 (uniCOIL vs. SPLADE) leads to more than 60% latency increase. In contrast, DLRs see lower retrieval latency, which appears to be insensitive to the average number of tokens per passage; for example, uniCOIL_{DLR} and DeLADE_{DLR} exhibit large differences in vector sparsity but have comparable latency when using vectors with the same dimensions. Even with different dimensions, our approach does not exhibit much variability in retrieval latency. We attribute this advantage to our two-stage retrieval approach, where computationally expensive end-to-end retrieval only relies on a few dimensions (see Section 5.4.3 for more details).

How does different slicing strategy impact the effectiveness of DLR? Finally, we examine three different slicing strategies on the wordpiece matching models.¹⁴ We densify lexical representations with 768 dimensions (i.e., $M = 768$). Table 5.11 reports results for the three lexical

¹⁴Recall that the vocabulary IDs for the whole word matching models are randomly assigned; thus, the slicing strategy is considered random.

Table 5.12: Effectiveness comparisons of DLRs with different unsupervised compression techniques on FiQA-2018 (Test).

Base model	Method	Dim	Quality		Storage
			nDCG@10 (diff.)	R@100 (diff.)	GB
DeLADE	FlatIP	30K	0.301	0.576	6.54
	LSH	-	0.210 (−30.2%)	0.545 (−5.4%)	0.75
	PQ768	-	0.281 (−6.6%)	0.557 (−3.3%)	0.07
	PQ256	-	0.253 (−15.9%)	0.535 (−7.1%)	0.04
	PQ128	-	0.221 (−26.6%)	0.496 (−13.9%)	0.03
		768	0.294 (−2.3%)	0.567 (−1.6%)	0.13
	DLR	256	0.287 (−4.7%)	0.558 (−3.1%)	0.04
		128	0.274 (−9.0%)	0.552 (−4.2%)	0.02

retrieval models, along with the number of tokens per passage for each condition. We notice that stride slicing has the same effectiveness as randomized slicing (only minor differences observed beyond four digits). However, surprisingly, the contiguous slicing strategy shows degradation in ranking effectiveness and the number of tokens per passage for this condition is smaller than the other slicing strategies. This indicates that BERT wordpiece tokens with adjacent token IDs may co-occur with higher probability than any two randomly chosen tokens. Thus, max pooling over contiguous slices of BERT wordpiece token IDs leads to more collisions compared to the other strategies. Based on this analysis, we use stride slicing as our default setting in the rest of our experiments. An optimized strategy is worth further exploring, which we leave for future work.

How well do DLRs compare to existing unsupervised vector compression algorithms? To compare our approach to the unsupervised vector compression techniques shown in Table 5.12, we use the FiQA-2018 test collection in BEIR [Thakur et al., 2021b], which is based on a medium-size corpus with 57K passages. Compared to MS MARCO (8.8M passages), it is less computationally demanding to perform unsupervised training for the various vector compression techniques. We use DeLADE as the base model and report results (nDCG@10 and R@100 as quality metrics and storage as the efficiency metric) in Table 5.12. Our experiments follow Faiss instructions¹⁵ for applying the unsupervised vector compression techniques. The FlatIP index (without any compression) provides the performance upper bound. For locality-sensitivity hashing (LSH), we use `faiss.IndexLSH` with `nbits = 8 · 768`; for product quantization (PQ), we use `faiss.IndexPQ` with `nbits = 8` and $M = 768, 256, 128$.¹⁶ We observe that PQ768

¹⁵<https://github.com/facebookresearch/faiss/wiki/Faiss-indexes>

¹⁶We also tried OPQ [Ge et al., 2014] training before performing PQ; however, the effectiveness is much worse.

can retain most of the information from the original high-dimensional vectors (less than 10% retrieval effectiveness drop) while LSH cannot effectively compress even a 30K-dimensional vector into binary codes. When we further compress the original vectors with PQ256 and PQ128, retrieval effectiveness drops more than 10% (and even more for PQ128). In contrast, DLR shows retrieval effectiveness drops less than 10% and performs consistently better than product quantization with the same storage size. Furthermore, our approach does not require any supervised or unsupervised training.

5.5 Summary

In this chapter, we present a solution to reduce the runtime cost of hybrid retrieval models, such as dense–sparse fusion and hybrid bi-encoder retrieval models. Specifically, we propose a dense representation framework and introduce a new scoring function to compute relevance scores between dense lexical representations (DLRs) derived from queries and passages. Using our framework, we can combine lexical and semantic representations into dense hybrid representations (DHRs) for hybrid retrieval. Our experiments show that DLRs can accurately approximate any “off-the-shelf” lexical model. Furthermore, when combined with other semantic representations (as DHRs), the resulting models can achieve effectiveness comparable to existing state-of-the-art hybrid retrieval methods.

The main advantage of our framework is that we can execute end-to-end retrieval using DLRs/DHRs on GPUs with a single index structure in a uniform execution environment. In our implementation, retrieval latency is insensitive to the sparsity of the vectors, unlike lexical representations that use inverted indexes. This feature makes our approach both fast and easy to deploy, especially for modern lexical retrieval models, which generate representations that are quite dense and require additional tuning (e.g., by introducing a sparsity constraint) to enable practical deployment using inverted indexes. Furthermore, we propose better modeling of content by jointly training semantic and lexical representations, which are then combined into hybrid representations for retrieval. We demonstrate that this combination, (DeLADE+[CLS])_{DHR}, outperforms most models (e.g., ANCE, ColBERT, COIL, etc.) in both in-domain and out-of-domain evaluations while requiring smaller indexes and achieving lower query latency. Finally, we examine our two-stage retrieval approach, revealing how the proposed design achieves low query latency without sacrificing accuracy.

Chapter 6

Aggregating Semantic and Lexical Features for Robust Dense Retrieval

6.1 Overview

Despite the success of dense retrieval [Reimers and Gurevych, 2019, Karpukhin et al., 2020], recent research has raised concerns about the robustness of these single-vector models [Thakur et al., 2021b, Sciavolino et al., 2021]. As shown in Fig.6.1, single-vector dense retrievers (e.g., BERT_{CLS} and TAS-B) trained with well-designed knowledge distillation strategies [Hofstätter et al., 2021] still underperform BM25 on out-of-domain datasets.

Recently, Gao and Callan [2021] observed that pre-trained language models such as BERT are not “structurally ready” for fine-tuning on downstream retrieval tasks. This is because the [CLS] token, pre-trained on the task of next sentence prediction (NSP), does not have the proper attention structure to aggregate fine-grained textual information. To address this issue, the authors propose further pre-training the [CLS] vector before fine-tuning and show that the gap between pre-training and fine-tuning tasks can be mitigated (see coCondenser_{CLS}, illustrated in Fig. 6.1). However, further pre-training introduces additional computational costs, which motivates us to ask the following question: Can we directly bridge the gap between pre-training and fine-tuning without any further pre-training?

Before diving into our proposed solution, we briefly overview the language modeling pre-training and DPR fine-tuning tasks using BERT. Fig.6.2(a) illustrates the BERT pre-training tasks: next sentence prediction (NSP) and masked language modeling (MLM), while Fig.6.2(b) shows the task of fine-tuning a dense retriever. We observe that relying solely on the [CLS]

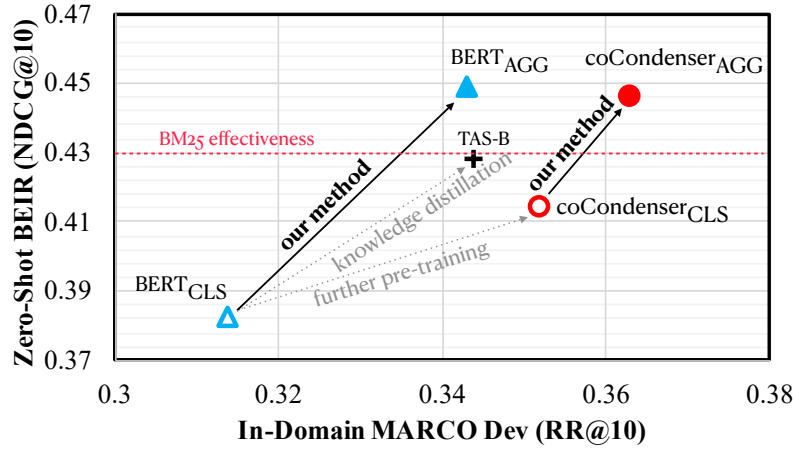


Figure 6.1: In-domain vs zero-shot effectiveness. All models are trained with the negatives mined by BM25.

vector as the dense representation does not exploit the full capacity of the pre-trained model, as the [CLS] vector is involved directly only in NSP during pre-training, and therefore lacks the information captured in the contextualized token embeddings. A simple solution is to aggregate the token embeddings by pooling (max or mean) into a single vector. However, information is lost in this process, and empirical results do not show consistent effectiveness gains. Hence, we recognize the need for better aggregation schemes.

Inspired by the single model fusion proposed in Section 5.2.3, in this chapter, we propose a novel approach to generate textual representations for retrieval that fully exploits contextualized token embeddings from BERT, as shown in Fig.6.2(c). Specifically, we reuse the pre-trained MLM head to map each contextualized token embedding into a high-dimensional wordpiece lexical space. Following a simple max-pooling and pruning strategy, we obtain a compact *lexical* vector, which we call agg^* . By concatenating agg^* and the [CLS] vector, our novel *Aggretriever* dense retrieval model captures representations pre-trained from both NSP and MLM, improving retrieval effectiveness by a noticeable margin compared to fine-tuned models that rely solely on the [CLS] vector (see BERT_{AGG} vs BERT_{CLS} in Fig.6.1). Note that, unlike the approach in Section 5.2.3, the final output vector of *Aggretriever* is a dense single vector. Thus, the scoring computation only involves the inner product between single vectors, rather than the gated inner product introduced in Section 5.2.1.

Importantly, fine-tuning *Aggretriever* does not require any sophisticated or computationally expensive techniques, making it a simple yet competitive baseline for dense retrieval. However, our approach is orthogonal to previously proposed further pre-training strategies and can still

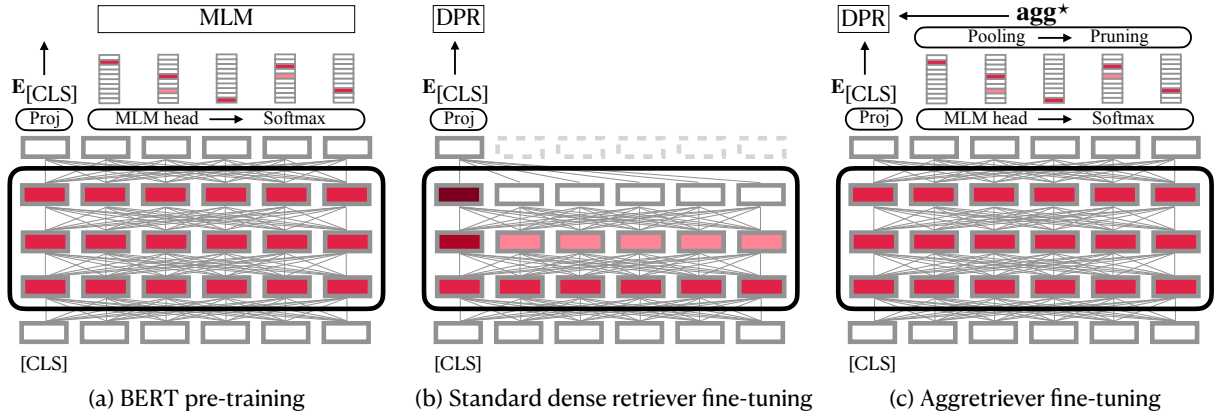


Figure 6.2: (a) BERT: next sentence prediction (NSP) and mask language modeling (MLM) (b) DPR: using the [CLS] embedding for retrieval (c) Aggretriever: aggregating knowledge from both NSP and MLM.

benefit from them to improve retrieval effectiveness even more (see $coCondenser_{AGG}$ in Fig. 6.1). To the best of our knowledge, this is the first work in the dense retrieval literature that leverages the pre-trained MLM head of BERT to encode textual information into a single dense vector.

6.2 Methodology: Aggretriever

In this section, we first introduce our method for text aggregation to form agg^* , which consists of two steps: pooling and pruning. Then, we describe how to concatenate the aggregated text representation agg^* and [CLS] into a 768-dimensional vector for fine-tuning and retrieval.

6.2.1 Text Aggregation Pooling

The goal of text aggregation is to transform contextualized token embeddings into a single-vector token representation. Let the input sequence q denote a tokenized query sequence with a length of l , $([CLS], q_1, \dots, q_l, [SEP])$, or a passage p of length m , $([CLS], p_1, \dots, p_m, [SEP])$. One simple approach is to directly pool (mean or max) contextualized token embeddings from the final layer. Such pooling strategies have been studied in previous work [Reimers and Gurevych, 2019], but do not appear to be consistently more effective than just using the [CLS] token; this is also confirmed in our ablation study (Section 6.4.5).

We instead propose to reuse the pre-trained MLM head to project each contextualized token embedding $\mathbf{e}_{q_i}$ into a high-dimensional vector in the wordpiece lexical space:

$$\mathbf{p}_{q_i} = \text{softmax}(\mathbf{e}_{q_i} \cdot \mathbf{W}_{\text{MLM}} + \mathbf{b}_{\text{MLM}}), \quad (6.1)$$

where $\mathbf{e}_{q_i} \in \mathbb{R}^d$, $\mathbf{W}_{\text{MLM}} \in \mathbb{R}^{d \times |\mathbf{V}_{\text{BERT}}|}$, and $\mathbf{b}_{\text{MLM}} \in \mathbb{R}^{|\mathbf{V}_{\text{BERT}}|}$ are the weights of the pre-trained MLM linear projector, and $\mathbf{p}_{q_i} \in \mathbb{R}^{|\mathbf{V}_{\text{BERT}}|}$ is the i -th contextualized token represented by a probability distribution over the 30522 tokens of BERT wordpiece vocabulary, $\mathbf{V}_{\text{BERT}}$. We then perform weighted max pooling for the sequential representations $(\mathbf{p}_{q_1}, \mathbf{p}_{q_2}, \dots, \mathbf{p}_{q_l})$ to obtain a single-vector lexical representation:

$$\mathbf{v}_q[v] = \max_{i \in \{1, 2, \dots, l\}} w_i \cdot \mathbf{p}_{q_i}[v], \quad (6.2)$$

where $w_i = |\mathbf{e}_{q_i} \cdot \mathbf{W} + \mathbf{b}| \in \mathbb{R}^1$ is a positive scalar and $v \in \{1, 2, \dots, |\mathbf{V}_{\text{BERT}}|\}$; $\mathbf{W} \in \mathbb{R}^{d \times 1}$ and $\mathbf{b} \in \mathbb{R}^1$ are trainable weights. Note that the scalar w_i for each token q_i is essential to capture term importance, which $\mathbf{p}_{q_i}$ alone cannot capture since it is normalized by softmax. We exclude the [CLS] token embedding at this stage since it is used for next-sentence prediction during pre-training and thus we argue that it does not carry much lexical information.

Our design has three advantages: (1) the MLM head with softmax is used for BERT pre-training; thus, the output probabilities can accurately model each contextualized token semantically. (2) In contrast to directly pooling contextualized embeddings, important dimensions of the token representations in the high-dimensional space are less likely to overlap, resulting in non-interfering max-pooling [Jang et al., 2021]. (3) Finally, w_i and $\mathbf{p}_{q_i}$ disentangle the effects of term importance from the MLM head. We will study the effectiveness of this design in Section 6.4.5 through ablations. Note that compared to previous work on sparse retrieval [Bai et al., 2020, Formal et al., 2021b], which switches softmax to ReLU to create sparse representations, our design sticks to the original activation function for MLM pre-training and directly outputs 30522-dimensional dense lexical vectors ($\mathbf{v}_q$).

Fig. 6.3(a) illustrates the generation of $\mathbf{v}_q$ with $|\mathbf{V}_{\text{BERT}}| = 10$ for simplicity. Ideally, we can directly compute $\langle \mathbf{v}_q, \mathbf{v}_p \rangle$ as a lexical matching similarity score for the wordpiece lexical representations. However, the vectors ($\mathbf{v}_q, \mathbf{v}_p \in \mathbb{R}^{|\mathbf{V}_{\text{BERT}}|}$) are too large for efficient retrieval using dense vector search libraries such as Faiss. To address this issue, we introduce our non-parametric pruning method to convert $\mathbf{v}_q$ ($\mathbf{v}_p$) into a low-dimensional vector for dense retrieval.

6.2.2 Text Aggregation Pruning

We consider $\mathbf{v}_q \in \mathbb{R}^{|\mathbf{V}_{\text{BERT}}|}$ as a bag-of-words representation with each dimension storing the corresponding term weight. Thus, dimensions with low term weights indicate that the corresponding

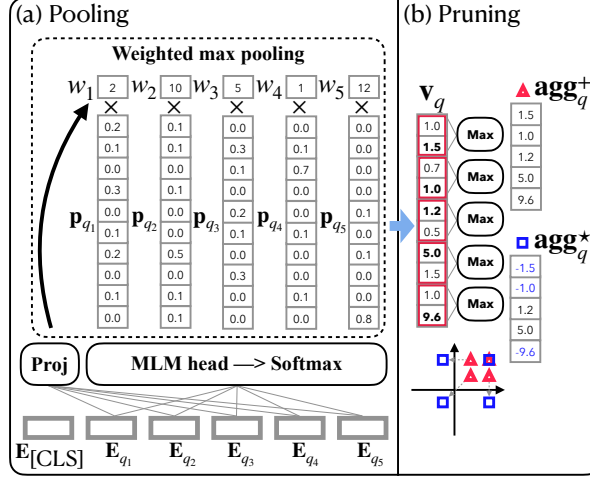


Figure 6.3: Illustration of text aggregation: (a) pooling of token representations to form $\mathbf{v}_q$; (b) pruning of $\mathbf{v}_q$ to form $\mathbf{agg}_q^*$ (or $\mathbf{agg}_q^+$). While pruning, $\mathbf{agg}_q^*[n]$ receives a negative value if the pooled element belongs to S_n^- ; i.e., the second element in each slice (red box).

terms are not important and can be pruned.

Based on this intuition, we propose to prune term weights in $\mathbf{v}_q$ by evenly and randomly dividing the dimensions (vocabulary) into d slices, $(S_1, S_2, \dots, S_d)$, where each slice consists of a set of $\frac{|V_{\text{BERT}}|}{d}$ index positions. Then, we condense $\mathbf{v}_q$ into a d -dimensional vector by pruning the term weights in each slice S_n :

$$\mathbf{agg}_q^+[n] = \max_{v \in S_n} \mathbf{v}_q[v]; \mathbf{id}_q[n] = \arg \max_{v \in S_n} \mathbf{v}_q[v]. \quad (6.3)$$

We call the operation in Eq. (6.3) *slice max pooling*, where each value in $\mathbf{agg}_q^+$ represents the weight of the most important term in the slice.¹ Slice max pooling is an important operation to prune the term weights while performing dimensionality reduction for dense passage retrieval. Other effective approaches to pruning lexical representations, e.g., top- k pruning [Yang et al., 2021] and FLOP regularization [Formal et al., 2021b], do not reduce the vector dimensionality. Thus, they generate sparse representation models are generated that require inverted indexes for efficient retrieval.

We call $\mathbf{agg}_q^+ \in \mathbb{R}^d$ the semi-aggregated lexical representation for query q since it only distributes vectors over the positive orthant and does not fully use the d -dimensional space. That

¹Slice *mean* pooling is less effective in our experiment.

is, $\mathbf{v}_q[v] \geq 0 \forall v \in \{1, 2, \dots, |\mathbf{V}_{\text{BERT}}|\}$; thus, $\mathbf{agg}_q^+[n] \geq 0 \forall n \in \{1, 2, \dots, d\}$. Our goal is to approximate the dot product between $\mathbf{v}_q$ and $\mathbf{v}_p$ in Eq. (6.2) by the ones in Eq. (6.3):

$$\langle \mathbf{v}_q, \mathbf{v}_p \rangle \approx \sum_{n=1}^d (\max_{v \in S_n} \mathbf{v}_q[v]) \cdot (\max_{v \in S_n} \mathbf{v}_p[v]) = \sum_{n=1}^d \mathbf{agg}_q^+[n] \cdot \mathbf{agg}_p^+[n] = \langle \mathbf{agg}_q^+, \mathbf{agg}_p^+ \rangle. \quad (6.4)$$

Note that the approximation error in Eq. (6.4) partially comes from *term misalignment*:

$$\mathbf{id}_q[n] \neq \mathbf{id}_p[n], \quad (6.5)$$

where the values in $\mathbf{agg}_q^+[n]$ and $\mathbf{agg}_p^+[n]$ do not represent the same term. Alternatively, this can be explained as fuzzy matching between two lexical representations since the two different wordpiece tokens may interact and contribute to the dot product. Term misalignment increases as d becomes smaller with respect to $|\mathbf{V}_{\text{BERT}}|$; thus, the error increases as well, which we show in Section 6.4.5.

To mitigate this error, we distribute the semi-aggregated lexical representation to the negative orthants to form what we call the fully aggregated lexical representation, distributed over the entire d -dimensional space.

$$\mathbf{agg}_q^*[n] = \begin{cases} \mathbf{agg}_q^+[n] & \text{if } \mathbf{id}_q[n] \in S_n^+ \\ -\mathbf{agg}_q^+[n] & \text{if } \mathbf{id}_q[n] \in S_n^-, \end{cases} \quad (6.6)$$

where S_n^+ and S_n^- are disjoint subsets of S_n (i.e., $S_n^+ \cup S_n^- = S_n$ and $S_n^+ \cap S_n^- = \emptyset$). That is, we evenly distribute the elements in S_n to S_n^+ and S_n^- .

The dot product between two fully aggregated lexical representations then becomes:

$$\text{sim}_{\text{agg}}(q, p) \triangleq \langle \mathbf{agg}_q^*, \mathbf{agg}_p^* \rangle = \sum_{n=1}^d \begin{cases} -\mathbf{agg}_q^+[n] \cdot \mathbf{agg}_p^+[n] & \text{if case (a) or (b)} \\ \mathbf{agg}_q^+[n] \cdot \mathbf{agg}_p^+[n] & \text{otherwise,} \end{cases} \quad (6.7)$$

where the cases are: (a) $\mathbf{id}_q[n] \in S_n^+$; $\mathbf{id}_p[n] \in S_n^-$; (b) $\mathbf{id}_q[n] \in S_n^-$; $\mathbf{id}_p[n] \in S_n^+$. That is, the dot product of $\mathbf{agg}^*$ in Eq. (6.7) avoids interactions between misaligned terms in the above cases (with 50% of probability), which $\mathbf{agg}^+$ in Eq. (6.4) does not consider. Note that we do not store the vectors $\mathbf{id}_p$ and $\mathbf{id}_q$ to compute Eq. (6.7). Fig. 6.3(b) illustrates the difference between $\mathbf{agg}_q^+$ and $\mathbf{agg}_q^*$ with $d = 5$, $|S_n| = 2$ and $|S_n^-| = |S_n^+| = 1$ for simplicity.

6.2.3 Fine-Tuning and Retrieval

Although $\mathbf{agg}^*$ can mitigate the issue of term misalignment, the approximation error cannot be completely eliminated unless $d = |\mathbf{V}_{\text{BERT}}|$. To enhance retrieval effectiveness, we concatenate the

agg^* vector with the $[\text{CLS}]$ vector since they are pre-trained to capture textual representations in different ways, focusing on the lexical and semantic, respectively.

In our Aggretriever model, the scoring function is the dot product of the concatenated vectors:

$$\text{sim}(q, p) \triangleq (\mathbf{e}_{q[\text{CLS}]} \oplus \text{agg}_q^*) \cdot (\mathbf{e}_{p[\text{CLS}]} \oplus \text{agg}_p^*),$$

where $\oplus$ means vector concatenation. The vector $\mathbf{e}_{q[\text{CLS}]} \oplus \text{agg}_q^*$ captures representations pre-trained from both NSP and MLM.

During fine-tuning, we aim to minimize the negative log-likelihood of a relevant query–passage pair. Specifically, given a query q , its relevant passage p^+ and a set of negative passages $\{p_1^-, p_2^-, \dots, p_{bs}^-\}$, we train our model by minimizing the NCE in Eq. (2.7):

$$-\log \frac{\exp(\text{sim}(q, p^+))}{\exp(\text{sim}(q, p^+)) + \sum_{j=1}^{bs} \exp(\text{sim}(q, p_j^-))}.$$

In addition, we use the same NLL loss, $\mathcal{L}_{\text{agg}}$ and $\mathcal{L}_{[\text{CLS}]}$, to optimize sim_{agg} and $\text{sim}_{[\text{CLS}]}$ separately. The final loss is as follows: $\mathcal{L} + \lambda_1 \cdot \mathcal{L}_{\text{agg}} + \lambda_2 \cdot \mathcal{L}_{[\text{CLS}]}$. We set λ_1 and λ_2 to 0.5 in all our experiments. While conducting retrieval, we use Flat IP in Faiss [Johnson et al., 2021] to index the passage vectors. Note that in our main experiments, we project $\mathbf{e}_{q[\text{CLS}]}$ and $\mathbf{e}_{p[\text{CLS}]}$ to 128 dimensions through a linear layer and set $d = 640$ for agg^* so that the dimensionality is 768.

6.3 Experimental Setups

6.3.1 Datasets

Supervised Evaluations. We evaluate supervised retrieval effectiveness on web search and open-domain question answering. For web search, we use the MS MARCO dataset aforementioned in Section 2.5. For open-domain question answering, we use the Wikipedia corpus released by Karpukhin et al. [2020], comprising 21M Wikipedia passages, and conduct experiments on two query sets, Natural Questions [NQ; Kwiatkowski et al., 2019] and Trivia QA [TQA; Joshi et al., 2017]. We directly use the training and test sets released by Karpukhin et al. [2020] for training and evaluation, respectively. That is, NQ consists of 58K (3.6K) training (test) queries while TQA comprises of 60K (11K) training (test) queries. For these tasks, we report hit accuracy at cutoffs 5, 20, and 100, denoted R@5/20/100.

Zero-Shot Evaluations. We evaluate zero-shot retrieval effectiveness on open-domain QA with two query sets, SQuAD [Rajpurkar et al., 2016] and EntityQuestions [EntityQs; Sciavolino et al., 2021], which are challenging for dense retrieval models. We report hit accuracy at cutoffs 20 and 100 (R@20/100). Additionally, we conduct evaluation on BEIR-14 mentioned in Section 2.5.

6.3.2 Models

Since our approach to text aggregation can be applied to any existing pre-trained encoder-only model, we test the effectiveness of Aggretriever on two pre-trained LM models and two further pre-trained models: (1) BERT [Devlin et al., 2019]; (2) DistilBERT [Sanh et al., 2019], a 6-layer transformer distilled from BERT; (3) Condenser [Gao and Callan, 2021], a BERT model further pre-trained with the tasks of auto-encoding and skip-connection MLM; and (4) coCondenser [Gao and Callan, 2022], a corpus-aware Condenser combining the tasks of skip-connection MLM and an ICT variant that comes in two separate flavors, further pre-trained on the MS MARCO and Wikipedia corpora, respectively. All model checkpoints can be downloaded from the Hugging Face Model Hub.² We compare models fine-tuned using only the [CLS] vector and based on our approach with the subscripts “CLS” and “AGG”, respectively, e.g., BERT_{CLS} and BERT_{AGG}. In addition, we also report the effectiveness of BM25 as a reference point; these results come from the Pyserini IR toolkit [Lin et al., 2021a].

6.3.3 Implementation Details

We implement our models using Tevatron [Gao et al., 2023] and apply its default training settings in most tasks. For MS MARCO, we train models for three epochs with a learning rate $5e - 6$, and for each batch, we include 8 queries. Each of the queries is paired with a randomly sampled positive passage and 7 negative passages mined using BM25. The maximum query and passage lengths are set to 32 and 128, respectively. Note that we use the official training set and corpus³ instead of the ones in Tevatron, which are further processed by Qu et al. [2021]. For open-domain QA, we follow the original settings used by Karpukhin et al. [2020] except for two modifications: (1) we use shared instead of independent weights between the query and passage encoders; (2) we set the maximum query and passage lengths to 32 and 156 for faster fine-tuning and inference. Note that we use one and four Tesla V100 GPUs (32GB) for model fine-tuning on MS MARCO

²<https://huggingface.co/models>

³<https://microsoft.github.io/msmarco/TREC-Deep-Learning-2019>

and open-domain QA, respectively. For BEIR evaluation, we use the APIs provided by Thakur et al. [2021b] and set maximum query and passage input lengths to 512.⁴

6.4 Experimental Results

6.4.1 Supervised Evaluations

Table 6.1: Supervised retrieval effectiveness comparisons. All models are fine-tuned with negatives from BM25. Bold denotes the best model for that metric.

Model	MARCO Dev		DL19 / 20	NQ Test			TQA Test		
	RR@10	R@1K	nDCG@10	R@5	R@20	R@100	R@5	R@20	R@100
(a) BM25	0.188	0.858	0.506 / 0.475	0.438	0.629	0.783	0.663	0.764	0.832
(1) DistilBERT _{CLS}	0.308	0.940	0.633 / 0.629	0.660	0.785	0.860	0.698	0.790	0.849
(2) DistilBERT _{AGG}	0.341	0.960	0.682 / 0.674	0.681	0.805	0.869	0.729	0.808	0.857
(3) BERT _{CLS}	0.314	0.942	0.612 / 0.643	0.677	0.799	0.863	0.710	0.796	0.852
(4) BERT _{AGG}	0.343	0.962	0.677 / 0.666	0.696	0.805	0.867	0.735	0.813	0.860
(5) Condenser _{CLS}	0.335	0.954	0.663 / 0.666	0.701	0.814	0.872	0.732	0.812	0.858
(6) Condenser _{AGG}	0.356	0.966	0.674 / 0.697	0.699	0.810	0.873	0.747	0.821	0.864
(7) coCondenser _{CLS}	0.352	0.973	0.674 / 0.684	0.707	0.818	0.878	0.745	0.819	0.867
(8) coCondenser _{AGG}	0.363	0.973	0.678 / 0.697	0.699	0.812	0.875	0.751	0.823	0.867

Fine-Tuning with Full Training Data. Table 6.1 compares in-domain retrieval effectiveness across the various models. We observe that our approach consistently improves on DistilBERT and BERT across all datasets, especially for metrics that emphasize top rankings. For example, DistilBERT_{AGG} sees a three-point and five-point improvement over DistilBERT_{CLS} on RR@10 and nDCG@10 for MS MARCO Dev and TREC DL, respectively, and over two points on R@5 for both NQ and TQA (row 2 vs 1). Similar trends can be observed on BERT (row 4 vs 3).

For the further pre-trained models, we observe that both Condenser_{AGG} and coCondenser_{AGG} yield effectiveness gains on MS MARCO and TQA (rows 6 and 8), which suggests that our approach is orthogonal and additive to further pre-training methods. We observe that in some cases, Aggretriever using pre-trained BERT as the backbone can obtain better retrieval effectiveness than further pre-trained models that are fine-tuned only on the [CLS] vector. For example, BERT_{AGG} outperforms Condenser_{CLS} for MS MARCO and TQA (row 4 vs 5). This indicates that existing language models pre-trained on MLM can serve as an effective single-vector dense

⁴<https://github.com/beir-cellar/beir>

Table 6.2: Near-domain zero-shot retrieval effectiveness comparisons using NQ or TQA for fine-tuning. Bold denotes the best model for that metric.

Target (Source)	SQuAD (NQ)		EntityQs (NQ)		SQuAD (TQA)		EntityQs (TQA)	
Model	R@20	R@100	R@20	R@100	R@20	R@100	R@20	R@100
(a) BM25	0.712	0.820	0.714	0.800	0.712	0.820	0.714	0.800
(1) DistilBERT _{CLS}	0.514	0.670	0.518	0.650	0.573	0.725	0.640	0.751
(2) DistilBERT _{AGG}	0.529	0.688	0.564	0.683	0.648	0.775	0.713	0.797
(3) BERT _{CLS}	0.512	0.671	0.534	0.664	0.581	0.722	0.637	0.747
(4) BERT _{AGG}	0.539	0.692	0.562	0.681	0.651	0.779	0.716	0.798
(5) Condenser _{CLS}	0.559	0.705	0.567	0.692	0.605	0.742	0.671	0.775
(6) Condenser _{AGG}	0.541	0.692	0.564	0.684	0.643	0.772	0.716	0.800
(7) coCondenser _{CLS}	0.567	0.715	0.556	0.684	0.629	0.762	0.695	0.791
(8) coCondenser _{AGG}	0.535	0.696	0.584	0.701	0.646	0.777	0.724	0.804

retriever, without further pre-training, using our proposed methods. Without corpus-aware further pre-training, Condenser_{AGG} is competitive with coCondenser_{CLS} on MS MARCO and TQA (row 6 vs 7).

6.4.2 Zero-Shot Evaluations

In these experiments, we examine robustness in a zero-shot retrieval setting. We first consider transfer to “near-domain” (Wikipedia) datasets, reported in Table 6.2. Specifically, we perform retrieval on test queries from SQuAD and EntityQs using models fine-tuned on NQ or TQA.

We see that Aggretriever with any backbone yields sizable gains over its [CLS] counterpart, with the exception that Condenser_{AGG} (and coCondenser_{AGG}) underperforms Condenser_{CLS} (and coCondenser_{CLS}) in SQuAD using NQ as the source (e.g., row 6 vs 5). It is worth mentioning that using TQA as the source, Aggretriever with any backbone is competitive with BM25 while the other [CLS] models still lag behind BM25 on the EntityQs test queries. Finally, we observe that models fine-tuned on TQA have better zero-shot retrieval effectiveness in near-domain datasets compared to those fine-tuned on NQ, which is also observed by Ram et al. [2022].

We evaluate zero-shot retrieval effectiveness on the multi-domain BEIR-14 dataset, reported in Table 6.3. We evaluate the models fine-tuned on three different sources: MS MARCO, NQ, and TQA. Similarly, Aggretriever shows better zero-shot retrieval effectiveness compared to its [CLS] counterpart with any backbone. For example, our model consistently and substantially outperforms the comparable baselines using MS MARCO and TQA as the source dataset for fine-tuning. Although models fine-tuned on NQ show the worst zero-shot retrieval capability,

Table 6.3: Multi-domain zero-shot retrieval effectiveness comparisons using various sources for fine-tuning. Bold denotes the best model for that metric.

Model	BEIR-14 (nDCG@10)		
(a) BM25	0.430		
Source	MARCO	NQ	TQA
(1) DistilBERT _{CLS}	0.364	0.262	0.266
(2) DistilBERT _{AGG}	0.450	0.277	0.386
(3) BERT _{CLS}	0.382	0.283	0.305
(4) BERT _{AGG}	0.449	0.299	0.394
(5) Condenser _{CLS}	0.393	0.286	0.314
(6) Condenser _{AGG}	0.447	0.295	0.385
(7) coCondenser _{CLS}	0.414	0.277	0.307
(8) coCondenser _{AGG}	0.446	0.280	0.376

Aggretriever with any backbone still slightly outperforms its [CLS] counterpart. It is also worth mentioning that Aggretriever with any backbone fine-tuned on MS MARCO outperforms the strong BM25 baseline.

6.4.3 Comparisons with State-of-the-Art dense retrieval models

Table 6.4 compares Aggretriever with existing dense retrievers fine-tuned with more expensive strategies; i.e., cross-encoder knowledge distillation (KD), hard negative mining (HNM), and large in-batch negatives, on both in-domain and out-of-domain evaluations. The two baseline models without further pre-training are: (1) TAS-B [Hofstätter et al., 2021], which distills ColBERT and a cross-encoder to DPR with an efficient topic-aware sampling strategy; (2) CL-DRD [Zeng et al., 2022], which further improves TAS-B by combining curriculum learning, hard negative mining, and cross-encoder knowledge distillation. Three models with further pre-training are included: (1) coCondenser [Gao and Callan, 2022], already discussed in Section 6.3.2; (2) Contriever [Izacard et al., 2022], which leverages pre-training by combining advanced contrastive learning techniques with an Inverse Cloze Task (ICT) variant; (3) GTR-Base [Ni et al., 2022], which trains a T5-Base encoder model that combines pre-training, knowledge distillation, and hard negative mining. For TAS-B, Contriever, and GTR-Base, we directly copy numbers from Izacard et al. [2022] and Ni et al. [2022], respectively. For CL-DRD⁵ and coCondenser,⁶ we use the models provided by the authors to conduct in-domain and out-of-domain evaluations ourselves. Note that the coCondenser model provided by the authors is fine-tuned

⁵<https://github.com/HansiZeng/CL-DRD>

⁶<https://huggingface.co/Luyu/co-condenser-marco-retriever>

Table 6.4: Comparisons with state-of-the-art dense retrieval models.

	w/o pre-training			w/ pre-training			
	<i>DistilBERT_{AGG}</i>	TAS-B	CL-DRD	<i>coCondenser_{AGG}</i>	<i>coCondenser</i>	Contriever	GTR-Base
knowledge distillation	✗	✓	✓	✗	✗	✗	✓
hard negative mining	✗	✗	✓	✗	✓	✓	✓
batch size > 1K	✗	✗	✗	✗	✗	✓	✓
MARCO	RR@10						
Dev	0.341	0.344	0.381	0.363	0.382*	0.341	0.366
BEIR	nDCG@10						
TREC-COVID	0.661	0.481	0.584	0.751	0.712	0.596	0.539
NFCorpus	0.297	0.319	0.315	0.323	0.325	0.328	0.308
NQ	0.474	0.463	0.500	0.490	0.487	0.498	0.495
HotpotQA	0.616	0.584	0.589	0.609	0.563	0.638	0.535
FiQA-2018	0.292	0.300	0.308	0.305	0.276	0.329	0.349
ArguAna	0.417	0.429	0.413	0.438	0.299	0.446	0.511
Tóuche-2020 (v2)	0.263	0.162	0.203	0.213	0.191	0.230	0.205
Quora	0.834	0.835	0.826	0.851	0.856	0.865	0.881
DBPedia	0.362	0.384	0.381	0.380	0.363	0.413	0.347
SCIDOCS	0.138	0.149	0.146	0.143	0.137	0.165	0.149
FEVER	0.781	0.700	0.734	0.600	0.495	0.758	0.660
Climate-FEVER	0.210	0.228	0.204	0.155	0.144	0.237	0.241
SciFact	0.630	0.643	0.621	0.650	0.615	0.677	0.600
CQADupStack	0.318	0.314	0.325	0.338	0.320	0.345	0.357
Avg.nDCG@10	0.450	0.428	0.439	0.446	0.413	0.466	0.441

* These numbers are not comparable due to the use of a “non-standard” MS MARCO passage corpus that has been augmented with title [Lassance and Clinchant, 2023].

in another round with self-mined hard negatives. Furthermore, they use a “non-standard” MS MARCO corpus where each passage is concatenated with a title; thus, the MS MARCO Dev results are different from the values for *coCondenser_{CLS}* reported in Table 6.1.

First, we observe that *DistilBERT_{AGG}* is not only competitive with TAS-B on in-domain evaluation but also outperforms both TAS-B and CL-DRD on out-of-domain evaluation, without needing supervision from an expensive cross-encoder teacher. Secondly, Contriever yields the best out-of-domain results at the cost of in-domain effectiveness. On the other hand, *coCondenser_{AGG}* reaches the same level of retrieval effectiveness as GTR-Base without leveraging any expensive fine-tuning strategies. Fine-tuning Aggretriever with knowledge distillation, hard negative mining, and large batch size is possible to further improve retrieval effectiveness, but these techniques

are orthogonal to our proposed model.

6.4.4 Ablations on Fine-Tuning Robustness

Table 6.5: Fine-tuning with limited data.

Model	MARCO Dev			
	RR@10		R@1K	
(a) BM25	0.188		0.858	
Train Size	1K	10K	1K	10K
(1) DistilBERT _{CLS}	0.145	0.222	0.754	0.865
(2) DistilBERT _{AGG}	0.207	0.260	0.868	0.905
(3) BERT _{CLS}	0.153	0.230	0.778	0.866
(4) BERT _{AGG}	0.207	0.258	0.871	0.906
(5) Condenser _{CLS}	0.191	0.259	0.841	0.903
(6) Condenser _{AGG}	0.211	0.258	0.873	0.899
(7) coCondenser _{CLS}	0.234	0.287	0.935	0.948
(8) coCondenser _{AGG}	0.209	0.280	0.880	0.914

Fine-Tuning with Limited Data. Table 6.5 reports retrieval effectiveness when the models are fine-tuned on subsets of the MS MARCO training data. Specifically, we randomly sample 1K and 10K queries from the training queries and fine-tune the models on each set for 40 epochs. We first observe that with only 1K training queries, both DistilBERT_{CLS} and BERT_{CLS} underperform BM25 (rows 1, 3 vs a), while both DistilBERT_{AGG} and BERT_{AGG} surpass BM25 (rows 2, 4 vs a) and are on par with Condenser_{CLS} (row 5), indicating that our approach successfully aggregates text information into a single vector without any further pre-training. We observe similar trends when fine-tuning models with 10K training queries. Finally, we find that coCondenser_{CLS} performs the best when fine-tuning with limited training data. This is probably because coCondenser’s further pre-training is designed for the [CLS] vector to learn corpus-aware signals from pseudo relevance in addition to skip-connection MLM. Thus, the [CLS] vector is more “ready” for retrieval with small training data.

Fine-Tuning with Noisy Hard Negatives. In this experiment, we use DistilBERT_{AGG} to examine Aggretriever’s robustness to fine-tuning with noisy hard negatives. Following TCT [Lin et al., 2021c] and RocketQA [Qu et al., 2021], for each query in the MS MARCO training set,

Table 6.6: Fine-tuning with noisy hard negatives.

Model	batch size	MARCO Dev	
		RR@10	R@1K
RocketQA [Qu et al., 2021]			
BM25 negatives	8K	0.333	-
+ noisy hard negatives	4K	0.260	-
+ denoising	4K	0.364	-
TCT-ColBERT [Lin et al., 2021c]			
BM25 negatives + knowledge distillation	96	0.344	0.967
+ noisy hard negatives	96	0.237	0.929
+ knowledge distillation	96	0.359	0.970
DistilBERT_{AGG}			
BM25 Neg.	64	0.341	0.960
+ noisy hard negatives	64	0.360	0.967

we retrieve the top-200 candidates using DistilBERT_{AGG} and further fine-tune the model by randomly sampling the candidates as negatives for two additional epochs using the same settings as the previous fine-tuning setup.

The results are listed in Table 6.6; we directly copy the numbers of TCT and RocketQA from the original papers. We notice that hard negatives reduce the effectiveness of both TCT and RocketQA since there are many false negatives in the candidates, as noted by Qu et al. [2021]. They address this issue using expensive training strategies: knowledge distillation (KD), denoising, and cross-batch negative sampling. On the other hand, DistilBERT_{AGG} obtains competitive retrieval effectiveness without any expensive training strategies. This experiment demonstrates that Aggretriever is robust and able to extract useful information when fine-tuned with noisy hard negatives.

6.4.5 Ablations on Text Aggregation

In this experiments, we use DistilBERT_{AGG} fine-tuned on the MS MARCO dataset to conduct an ablation study. In addition to MARCO Dev, to understand the zero-shot effectiveness of each condition, we conduct retrieval on a subset of BEIR (denoted BEIR small), consisting of five datasets from different domains: NFCorpus, FiQA, ArguAna, SCIDOCS, and SciFact. We report nDCG@10 averaged over these five datasets.

Table 6.7: DistilBERT_{AGG} text aggregation ablation. We project [CLS] to 128 dimensions and concatenate with a 640-dimensional embedding pooled and pruned using different strategies. AVERAGE denotes average pooling over all 768-dimensional contextualized token embeddings other than [CLS].

	Pooling		Pruning	MARCO Dev		BEIR small
	MLM	Weight		RR@10	R@1K	nDCG@10
(1)	✓	✓	full aggregation	0.341	0.960	0.355
(2)	✓	✗	full aggregation	0.308	0.937	0.308
(3)	✗	✓	full aggregation	0.332	0.953	0.355
(4)	✓	✓	semi aggregation	0.341	0.960	0.322
(5)	✓	✓	$\text{linear}(V_{\text{BERT}} \rightarrow 640)$	0.327	0.959	0.313
(6)	AVERAGE		$\text{linear}(768 \rightarrow 640)$	0.300	0.933	0.270
(7)	RepBERT [Zhan et al., 2020b]		-	0.306	0.942	0.264

Pooling Stage Ablation. In the second ablation experiment, we fix [CLS] and agg^* to 128 and 640 dimensions, respectively, and compare different designs of the pooling stage to form agg^* , as discussed in Section 6.2.1. The results are reported in the first main block of Table 6.7; row 1 is our default condition. In row 2, we remove the term importance component and assign a term weight of one for weighted max pooling. A substantial drop in retrieval effectiveness can be observed. In row 3, we remove MLM projection and represent each query (or passage) token with the 30522-dimensional indicator vector in Eq. (6.1); that is, $\mathbf{p}_{q_i} = x_j \in \{0, 1\}^{|V_{\text{BERT}}|}$ for $j \in \{\text{token.id}(q_i)\}$. We notice that skipping the MLM projector modestly harms retrieval effectiveness. This means that most textual information can be captured without the MLM projector, but it *does* help. This is sensible since the 30522-dimensional indicator vector still retains each original query (or passage) term. A comparison of row 2 and row 3 shows that learned term weights for each token are more important than the term semantic distribution (projected by MLM) over the wordpiece vocabulary.

Pruning Stage Ablation. In the second main block of Table 6.7, we study the effects of pruning wordpiece lexical representations on Aggretriever. For example, we semi-aggregate (linearly project) the lexical representations into 640-dimensional dense vectors, as shown in row 4 (5). We observe that our non-parametric pruning approaches are better than the learned ones (rows 1, 4 vs 5). Although agg^+ shows the same retrieval effectiveness as agg^* on in-domain evaluation, a substantial drop can be observed on out-of-domain evaluation (row 1 vs 4). This result demonstrates that our fully aggregated representations better preserve information from lexical representations and appear to be more robust to domain shifts.

We observe that projecting averaged contextualized embedding (excluding the [CLS]), denoted AVERAGE, into 640 dimensions, and then concatenating with [CLS] (row 6), does

Table 6.8: DistilBERT_{AGG} dimensionality ablation.

	Dim.		MARCO Dev		BEIR small
	[CLS]	agg*	RR@10	R@1K	nDCG@10
(1)	768	0	0.308	0.940	0.259
(2)	640	128	0.327	0.954	0.307
(3)	128	640	0.341	0.960	0.355
(4)	0	768	0.307	0.926	0.328
(5)	768	768	0.350	0.966	0.358
(6)	128	128	0.320	0.946	0.300
(7)	0	30522	0.345	0.956	0.363

not perform well, indicating that projecting contextualized token embeddings into the high-dimensional wordpiece lexical space before pooling is key to preserving lexical information. Finally, we also try average pooling over all contextualized embeddings (including the [CLS]), which corresponds to RepBERT [Zhan et al., 2020b]. This yields negligible effectiveness difference from AVERAGE concatenated with [CLS] (row 7 vs 6); i.e., 0.306 (RR@10) and 0.264 (nDCG@10) on MARCO Dev and BEIR small, respectively.

Dimensionality Ablation. We first study the effects of dimensionality on the [CLS] and agg* vectors in Table 6.8. We find that [CLS] alone slightly outperforms agg* alone (row 1 vs 4) on in-domain evaluation while the reverse trend is seen on zero-shot evaluation. This observation indicates that the [CLS] and agg* vectors encode text in different ways and that combining them further improves retrieval effectiveness (row 5). Compared to [CLS] alone and agg* alone, we still see a slight improvement for in-domain evaluation at 256 dimensions (row 6 vs 1 and 4). Holding the number of dimensions constant (rows 1–4), the best condition (row 3) indicates that the agg* vector requires more space than the [CLS] vector.

Finally, we report the retrieval effectiveness of the original wordpiece lexical representations before pruning (row 7), which can be considered the effectiveness upper bound of agg*. Although agg* with 768 dimensions has lower effectiveness (row 4 vs 7), combined with [CLS], Aggretriever reduces the gap (rows 3, 5 vs 7), with better retrieval efficiency in terms of smaller index size and lower retrieval latency. For example, on the MS MARCO dataset, representing each passage as a 768-dimensional vector in a Faiss Flat index with 32 (16) bits requires 26 (13) GB and 100 ms/q retrieval latency on a single V100 GPU, while the 30522-dimensional vectors (without pruning) require around 40 times more index storage and are not practical for end-to-end retrieval.

Table 6.9: Query encoding latency comparisons.

		latency (1 st / 50 th / 99 th perc.)	
		CPU	GPU
(1)	DistilBERT _{CLS}	93 / 103 / 122 ms	15 / 16 / 18 ms
(2)	DistilBERT _{AGG}	155 / 163 / 191 ms	18 / 19 / 24 ms
(3)	w/o MLM	103 / 109 / 138 ms	16 / 19 / 20 ms

6.4.6 Query Encoding Latency

Although different single-vector dense retrievers with the same vector dimensionality have similar retrieval latency under the same software and environment when performing top- k retrieval, query encoding latency is also an important component to consider. In this experiment, we compare the query encoding latency of DistilBERT_{AGG} and DistilBERT_{CLS}. We measure the time required to encode the 6980 queries from MARCO Dev with batch size one on the CPU and GPU, using one thread on a Linux machine with a 2.2 GHz Intel Xeon Silver 4210 CPU and a single Tesla V100 GPU (32GB), respectively. We report the latency at 1th, 50th and 99th percentiles in Table 6.9.

We observe that query encoding with Aggretriever is slightly slower than its [CLS] counterpart on the GPU (row 2 vs 1). On the CPU, the gap is much larger, especially for tail queries. However, from row 3 (the same condition as row 3 in Table 6.7), we see that skipping the MLM head projection step reduces the query encoding latency with only a small retrieval effectiveness loss. For a real-world application, this might be a sensible option, bringing query encoding latency roughly in line with the [CLS]-only model.

6.5 Summary

In this chapter, we present Aggretriever, a single-vector dense retrieval model that fully exploits both lexical and semantic features extracted from a pre-trained transformer. We introduce a simple approach to aggregate the high-dimensional lexical representations extracted from MLM prediction into a dense vector, **agg***. Experiments show that **agg***, when combined with the extracted semantic vector, [CLS], achieves better retrieval effectiveness than using the [CLS] vector alone in both supervised and zero-shot evaluations. Our work demonstrates that the newly proposed dense retrieval model design can reduce training costs while yielding better retrieval effectiveness compared to existing dense retrieval models that rely on hard negative mining and knowledge distillation.

Chapter 7

A Simple Yet Effective Approach to Zero-Shot Multilingual Dense Retrieval

7.1 Overview

Fine-tuning a pre-trained transformer has proven highly effective for many natural language processing tasks, including information retrieval (IR). Despite its success, recent state-of-the-art dense retrieval models [Ni et al., 2022, Lin et al., 2023] predominantly focus on English. This bias arises because constructing an effective dense retrieval model requires a substantial amount of annotated training data, which is primarily available in English datasets [Bajaj et al., 2016, Kwiatkowski et al., 2019]. As a result, users of low-resource languages face challenges in benefiting from recent IR advancements.

To address this issue, researchers have explored leveraging pre-trained multilingual transformers, such as mBERT [Devlin et al., 2019] and XLM-R [Conneau et al., 2020], which exhibit promising language transferability even when fine-tuned on English datasets alone [Zhang et al., 2023b]. However, Izacard et al. [2022] emphasize the importance of contrastive pre-training on multilingual corpora to achieve improved language transferability in information retrieval (IR). Other approaches utilize multilingual parallel corpora or translation pairs for contrastive pre-training [Feng et al., 2022] or fine-tuning [Reimers and Gurevych, 2020, Bonifacio et al., 2021]. However, these solutions are only viable when significant computational resources or multilingual parallel data are available.

In Chapter 6, we demonstrate that existing dense retrieval models that solely use [CLS] (or averaged pooling) do not fully exploit the capabilities of pre-trained transformers for retrieval.

We propose Aggretriever by combining the semantic and lexical features from the two respective components, [CLS] and masked language modeling (MLM) prediction, which shows superior effectiveness on diverse English retrieval tasks. This leads us to ask the question: Can Aggretriever benefit multilingual retrieval?¹

Extending Aggretriever to support multilingual retrieval poses challenges due to the increased computational costs of extracting lexical features from the MLM component as the vocabulary size of the pre-trained models grows, making fine-tuning with limited resources challenging. In this work, we propose two simple approaches to approximate MLM prediction for extracting lexical features from multilingual pre-trained transformers, enabling the training of mAggretriever on a single GPU with 40 GB of memory. Remarkably, mAggretriever exhibits strong retrieval capabilities across multiple languages despite being fine-tuned solely on English data.

7.2 Methodology: mAggretriever

We extend Aggretriever to multilingual retrieval, mAggretriever. However, directly applying Eq. (6.1) to multilingual pre-trained MLM heads over the whole vocabulary increases computation cost for both training and inference due to large vocabulary size $|V_{wp}|$ in MLM projector, $\mathbf{W}_{mlm}$ [Nair et al., 2022, Lassance, 2023]. For example, the mBERT and XLM-R have respective vocabulary size of 120K and 250K (vs BERT’s 35K) in the MLM projector. To address the issue, we propose two computationally efficient approaches to approximate MLM prediction in Eq. (6.1).

Target-language prediction. One intuitive approach is to compute the probability distribution over the target-language token set of interest, denoted as V_{wp}^{tg} , by replacing $\mathbf{W}_{mlm}$ and $\mathbf{b}_{mlm}$ in Eq. (6.1) with $\mathbf{W}_{mlm}^{tg}$ and $\mathbf{b}_{mlm}^{tg}$, respectively.

$$\begin{cases} \mathbf{W}_{mlm}^{tg}[:, v] = \mathbf{W}_{mlm}[:, v]; \\ \mathbf{b}_{mlm}^{tg}[v] = \mathbf{b}_{mlm}[v], & \text{if } v \in V_{wp}^{tg} \\ \mathbf{W}_{mlm}^{tg}[:, v] = \vec{0}; \mathbf{b}_{mlm}^{tg}[v] = 0, & \text{otherwise.} \end{cases}$$

From the above equation, we only have to compute the matrix multiplication and softmax among $|V_{wp}^{tg}|$ instead of $|V_{wp}|$ tokens. This approach assumes that only the tokens corresponding to the target language have responses when conducting MLM prediction; i.e., $\mathbf{p}_{q_i}[v] = 0$ if $v \notin V_{wp}^{tg}$.

¹In this chapter, we refer to multilingual retrieval as monolingual retrieval across multiple languages.

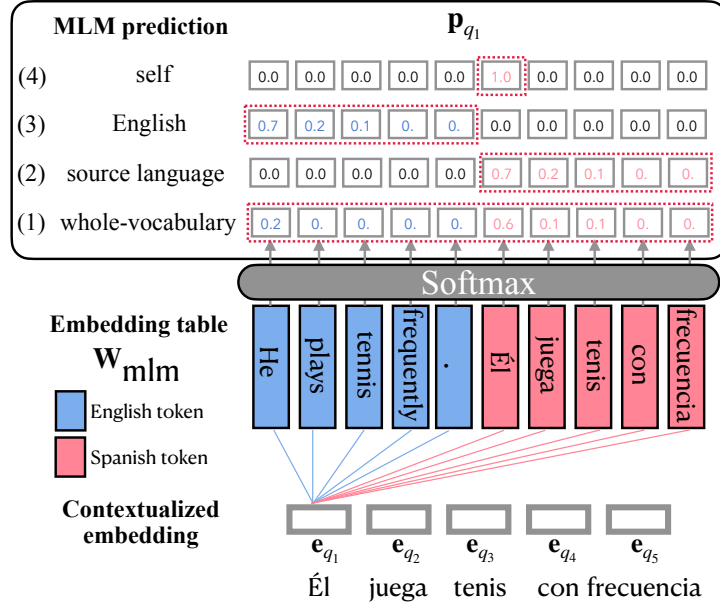


Figure 7.1: Illustration of MLM prediction for the input token q_1 . (1) Whole-vocabulary prediction considers all languages while target-language prediction only considers the tokens in one language; e.g., (2) source language (Spanish) or (3) English. (4) Self prediction assigns probability of one to itself. 0. denotes less than 0.1.

Self prediction. Since BERT MLM pre-training task includes recovering the unmasked token itself, to further save computation cost, we may approximate MLM prediction by assigning a value of one to the token q_i itself, and zero to the others:

$$\mathbf{p}_{q_i} = x_j \in \{0, 1\}^{|\mathbf{V}_{\text{wp}}|} \text{ for } j \in \{\text{tok_id}(q_i)\}. \quad (7.1)$$

The operation removes the computationally expensive matrix multiplication and softmax operation in Eq. (6.1). Note that, combining Eq. (6.2) and (7.1), the lexical representations, $\mathbf{v}_q$, from self prediction can be considered bag-of-word vectors with learned term weights [Lin and Ma, 2021].

Figure 7.1 illustrates whole-vocabulary, our proposed target-language and self MLM predictions. Table 7.1 compares the training and inference cost of the standard dense retrieval (row 1) and mAggretriever with different MLM prediction strategies on MS MARCO dataset using the backbone of XLM-R base. The training and query encoding settings are detailed in Section 7.3.2. We observe that training with target-language (English in our case) MLM prediction reduces the GPU memory requirement compared to whole vocabulary (row 3 vs 2) while self

Table 7.1: Training and inference cost comparisons on MS MARCO with XLM-R base.

Models	vector components		MLM prediction	training cost		query encoding latency	
	[CLS]	agg*		GPU memory	total time	CPU	GPU
(1) XLM-R _{CLS}	768 dim.	0 dim.	-	21 GBs	5.5 hrs	105 ms/q	9.1 ms/q
(2) XLM-R _{AGG}	128 dim.	640 dim.	whole vocabulary	OOM (140 GBs)	-	400 ms/q	15.1 ms/q
(3) XLM-R _{AGG} ^{lg}			target language	39 GBs	9.5 hrs	248 ms/q	11.4 ms/q
(4) XLM-R _{AGG} ^{self}			self	21 GBs	5.5 hrs	123 ms/q	9.1 ms/q

prediction yields training and inference efficiency on par with standard dense retrieval (row 4 vs 1).

7.3 Experimental Setups

7.3.1 Datasets and Metrics

English Training Data. We use the MS MARCO passage ranking dataset introduced by Bajaj et al. [2016], comprising a corpus with 8.8M passages and 500K training queries. Model supervised language (English) retrieval effectiveness is evaluated on the 6980 (MARCO Dev) queries from the development set with one relevant passage per query on average. Following the established procedure, we report MRR@10 and R@1000 as the metrics.

Multilingual Testing Data. We evaluate model zero-shot retrieval effectiveness in other languages using MIRACL dataset introduced by Zhang et al. [2023c], comprising around 77k queries over Wikipedia in 18 languages with over 700k high-quality relevance judgments by native speakers. We use the publicly available development queries and their relevance judgments in 15 languages (two surprise languages and English are excluded).² Following Zhang et al. [2023c], we list the full numbers and report macro averaged nDCG10 and R@100 over the 15 languages.

Finally, we study how to conduct cross-lingual retrieval using mAggretriever on XQuAD-R dataset introduced by Roy et al. [2020], consisting of parallel queries and corpora with 11 languages.³ We conduct retrieval using the queries with X_Q language against the corpus with X_C language and report the macro-averaged MAP@100 over all the cross-lingual combinations of the 110 language pairs ($X_Q \neq X_C$), and the other 11 monolingual combinations ($X_Q = X_C$).

²<https://huggingface.co/datasets/miracl/miracl>

³<https://github.com/google-research-datasets/lareqa>

Table 7.2: MIRACL and XQuAD-R data statistics.

language	ISO	MIRACL Dev		XQuAD-R	
		# queries	# candidates	# queries	# candidates
Arabic	ar	2,869	2,061,414	1,190	1,222
Bengali	bn	411	297,265	-	-
German	de	-	-	1,190	1,276
Greek	el	-	-	1,190	1,234
English	en	648	32,893,221	1,190	1,180
Spanish	es	799	10,373,953	1,190	1,215
Persian	fa	632	2,207,172	-	-
Finnish	fi	1,271	1,883,509	-	-
French	fr	343	14,636,953	-	-
Hindi	hi	350	506,264	1,190	1,244
Indonesian	id	960	1,446,315	-	-
Japanese	ja	860	6,953,614	-	-
Korean	ko	213	1,486,752	-	-
Russian	ru	1,252	9,543,918	1,190	1,219
Swahili	sw	482	131,924	-	-
Telugu	te	828	518,079	-	-
Thai	th	733	542,166	1,190	852
Turkish	tr	-	-	1,190	1,167
Vietnamese	vi	-	-	1,190	1,209
Chinese	zh	393	4,934,368	1,190	1,196

Table 7.2 reports the data statistics of MIRACL and XQuAD-R. Note that the candidates of MIRACL are passages while the candidates of XQuAD-R are chunked sentences from XQuAD corpora. Although XQuAD corpora is manually rewritten multilingual parallel corpora [Artetxe et al., 2020], the numbers of chunked sentences are different between languages. Also note that for each query, MIRACL has multiple relevant candidates while XQuAD only has one.

7.3.2 Implementation Details

Models. We apply mAggretriever to two 12-layer pre-trained multilingual models: (1) mBERT;⁴ (2) XLM-R.⁵ We compare models fine-tuned solely using [CLS] vector and based on mAggretriever using whole-vocabulary MLM prediction with the subscripts “CLS” and “AGG”, respectively, e.g., mBERT_{CLS} and mBERT_{AGG}. We report the two variants of mAggretriever with target-language and self prediction; e.g., mBERT_{AGG}^{tg} and mBERT_{AGG}^{self}. For target-language prediction, we train mAggretriever using English token prediction and run inference using the corresponding language of each corpus, and for whole-vocabulary prediction, we fine-tune with half of the batch size. In addition, we report the numbers of BM25 and mDPR from Zhang et al. [2023c]

⁴<https://huggingface.co/bert-base-multilingual-cased>

⁵<https://huggingface.co/xlm-roberta-base>

as reference points, and the two state-of-the-art multilingual retrievers: (1) mContriever [Izacard et al., 2022], pre-trained on multilingual corpora with 29 languages and further fine-tuned on MS MARCO dataset;⁶ (2) Cohere (API), whose numbers are copied from Kamalloo et al. [2023].⁷

Training and Inference. We train our models on a single A100 GPU with 80 GB memory for 6 epochs (around 100k steps) with learning rate $7e-6$. Each batch includes 24 queries, and for each query, we randomly sample one positive and seven negative passages. All the negatives are sampled from the MS MARCO “small” triples training set, which is created using BM25. During training, we minimize the negative log likelihood as the standard contrastive loss and following Karpukhin et al. [2020], for each query, we consider all the (positive and negative) passages from the other triplets in the batch as in-batch negative samples. We set the maximum input length for the query and the passage to 32 and 128, respectively, at both training and inference stages for MS MARCO. For MIRACL and XQUAD-R, we use the maximum input length of 128 and 256 for the query and passage, respectively. Since our preliminary experiments on MS MARCO show that lowercase improves mAggretriever while degrades its CLS counterpart, we lowercase all the queries and passages for mAggretriever. We measure query encoding latency on the 6980 MS MARCO development queries with the batch size of 1 and single thread on a Linux machine with 12 Intel(R) Xeon(R) Gold 5317 CPU @ 3.00GHz and 88G of RAM.

Target Token Set Construction. For each language of corpus in MIRACL and XQuAD-R, we tokenize and lowercase all the passages and collect the unique tokens in the corpus as the target token set. For example, when fine-tuning on MS MARCO dataset, we use the token set built from MS MARCO corpus. While conducting target-language MLM prediction on MIRACL Arabic queries and corpus, we use the token set collected from Arabic corpus as our target token set. Note that self MLM prediction does not require collecting the token set for the target language.

7.4 Experimental Results

Evaluations on MIRACL. Table 7.3 reports models’ retrieval effectiveness on MS MARCO and MIRACL development queries. We first observe that mAggretriever, incorporating lexical features, not only outperforms its CLS counterpart in supervised English retrieval (MARCO), but also exhibits superior transferability to other languages (MIRACL) regardless of backbone. In addition, mBERT_{AGG}, without introducing contrastive pre-training on multilingual corpora,

⁶<https://huggingface.co/facebook/mcontriever-msmarco>

⁷<https://docs.cohere.com/docs/multilingual-language-models>

Table 7.3: Supervised (MARCO Dev) English and zero-shot (MIRACL Dev) multilingual retrieval effectiveness comparisons. The best effectiveness in each column is highlighted in bold.

MARCO Dev				MIRACL Dev													
Models	en	ar	bn	es	fa	fi	fr	hi	id	ja	ko	ru	sw	te	th	zh	avg.
	MRR@10	nDCG@10															
(a) BM25	18.8	48.1	50.8	31.9	33.3	55.1	18.3	45.3	44.9	36.9	41.9	33.4	38.3	49.4	48.4	18.0	39.6
(b) mDPR	29.6	49.9	44.3	47.8	48.0	47.2	43.5	38.3	27.2	43.9	41.9	40.7	29.9	35.6	35.8	51.2	41.7
(c) mContriever	27.4	52.5	50.0	41.8	21.5	60.2	31.4	28.6	39.2	42.4	48.3	39.1	56.0	52.8	51.7	41.0	43.8
(d) Cohere (API)	-	61.7	59.4	23.3	47.1	63.4	46.2	49.3	44.6	46.0	49.6	46.9	61.1	61.3	54.6	36.5	50.1
(1) mBERT _{CLS}	29.1	50.9	45.2	31.7	30.5	48.5	30.5	37.7	19.8	43.4	40.0	27.0	21.2	36.4	37.7	43.9	36.1
(2) mBERT _{AGG}	34.3	56.9	48.5	44.0	43.1	63.4	40.8	34.1	40.3	49.3	43.9	44.9	44.6	44.7	27.2	40.2	44.4
(3) mBERT _{AGG} ^{tg}	34.5	55.7	45.5	45.4	42.4	62.3	42.9	34.3	38.7	47.7	43.4	45.2	45.5	43.9	32.5	41.5	44.4
(4) mBERT _{AGG} ^{self}	34.2	59.4	51.0	44.6	44.5	65.3	43.6	37.4	42.1	50.2	47.8	46.3	48.5	48.0	31.1	44.3	46.9
(5) XLM-R _{CLS}	31.1	46.6	46.6	29.9	43.5	44.4	28.5	41.7	31.7	40.7	45.6	27.5	22.5	50.5	53.9	35.9	39.3
(6) XLM-R _{AGG}	34.7	60.6	60.4	42.5	46.2	66.2	43.0	44.8	47.8	53.3	58.2	44.6	46.6	72.1	66.0	41.4	52.9
(7) XLM-R _{AGG} ^{tg}	35.0	60.2	60.9	44.3	46.7	65.0	43.8	48.0	47.2	53.3	59.1	45.8	45.2	70.2	67.1	42.9	53.3
(8) XLM-R _{AGG} ^{self}	35.0	61.4	61.4	42.9	46.5	66.2	41.3	46.2	48.4	53.9	57.9	46.5	47.5	71.2	66.8	41.0	53.3
	R@1K	R@100															
(a) BM25	85.8	88.9	90.9	70.2	73.1	89.1	65.3	86.8	90.4	80.5	78.3	66.1	70.1	83.1	88.7	56.0	78.5
(b) mDPR	94.6	84.1	81.9	86.4	89.8	78.8	91.5	77.6	57.3	82.5	73.7	79.7	61.6	76.2	67.8	94.4	78.9
(c) mContriever	97.1	92.5	92.1	84.1	65.4	95.3	82.4	64.6	80.2	87.8	87.5	85.0	91.1	96.1	93.6	90.3	85.9
(d) Cohere (API)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(1) mBERT _{CLS}	93.6	84.3	81.9	67.1	84.1	61.2	72.9	76.3	47.7	80.7	71.9	61.1	50.4	75.5	69.7	90.0	71.6
(2) mBERT _{AGG}	95.8	86.9	83.8	81.0	81.0	90.2	83.2	70.9	76.6	84.1	71.7	80.8	79.4	82.4	60.0	81.3	79.5
(3) mBERT _{AGG} ^{tg}	96.1	86.2	82.2	82.1	79.9	89.7	82.9	72.6	75.7	84.3	76.3	80.5	80.9	81.7	63.8	82.6	80.1
(4) mBERT _{AGG} ^{self}	95.6	88.7	85.1	80.1	82.4	92.1	86.5	77.6	80.2	85.5	78.4	82.0	81.5	83.4	65.9	84.5	82.3
(5) XLM-R _{CLS}	93.8	79.2	82.6	63.6	79.9	75.2	66.8	76.9	62.7	77.2	77.6	62.5	48.8	87.0	89.2	78.9	73.9
(6) XLM-R _{AGG}	96.1	89.8	92.5	78.8	85.4	92.6	81.8	85.1	83.5	89.0	88.5	78.9	79.1	95.4	95.1	80.6	86.4
(7) XLM-R _{AGG} ^{tg}	96.2	89.3	93.1	80.4	84.6	91.8	81.4	84.5	82.6	88.9	86.6	79.5	77.7	94.8	95.1	79.9	86.0
(8) XLM-R _{AGG} ^{self}	96.0	90.2	92.8	78.2	84.5	93.1	81.2	82.5	84.6	89.8	88.3	80.8	79.4	95.5	94.5	79.0	86.3

outperforms mContriever in MIRACL in terms of nDCG@10. Note that mBERT_{AGG} and mContriever are both initialized from mBERT. We hypothesize that mContriever’s high recall (i.e., R@1K) comes from its pre-training on multilingual corpora with 29 languages.

Switching to XLM-R backbone, mAggretriever even outperforms Cohere (API). Furthermore, we notice that mAggretriever with XLM-R backbone improves over mBERT more than its CLS counterpart does. For example, in the case of MIRACL, XLM-R_{AGG}^{tg} exhibits a significant improvement over mBERT_{AGG}^{tg} from an nDCG@10 score of 44.4 to 53.3, whereas XLM-R_{CLS} only sees a modest improvement over mBERT_{CLS} from 36.1 to 39.3. This notable enhancement highlights mAggretriever’s ability to effectively utilize a superior pre-trained language model.

Finally, compared to whole-vocabulary and proposed approximate MLM predictions, we observe that self prediction shows comparable and even strong language transferability. We hypothesize that MLM prediction learned from English data cannot transfer well to other languages. It is worth mentioning that compared to whole-vocabulary MLM prediction, the proposed approximate MLM prediction, target-language and self prediction, are advantageous for real-world

Table 7.4: Zero-shot retrieval effectiveness on XQuAD-R.

Models	MLM target lang.		XQuAD-R	
	query	corpus	$X_Q = X_C$	$X_Q \neq X_C$
			MAP@100	
(1) XLM-R _{CLS}	-	-	73.1	57.5
(2) XLM-R _{AGG}	-	-	77.4	41.8
(3)	X_Q^*	X_C^*	77.4	36.0
(4) XLM-R _{AGG} ^{tg}	X_C^*	X_C^*	77.4	44.9
(5)	English	English	73.5	51.7
(6) XLM-R _{AGG} ^{self}	-	-	77.3	36.2

* X_Q (X_C) denotes the language of queries (corpus).

deployment since they show almost no effectiveness drop (sometimes even better) but require far less training and inference cost as shown in Table 7.1.

Evaluations on XQuAD-R. In Table 7.4, we directly apply XLM-R based models fine-tuned on MS MARCO to XQuAD-R dataset. In the experiment, we try different MLM prediction settings for XLM-R_{AGG}^{tg}. For example, instead of using respective query and corpus source language as the target language (row 3), we use the language corresponding to each corpus (row 4) or English (row 5) as target language for both queries and corpus. Note that rows 3 and 4 are the same when both queries and corpus are in the same language ($X_Q = X_C$).

We observe that mAggretriever shows relatively poor cross-lingual retrieval effectiveness ($X_Q \neq X_C$) compared to its CLS counterpart (rows 2,3,6 vs 1). When aligning the MLM prediction target language for queries and corpus, the cross-lingual retrieval effectiveness sees improvement (rows 3 vs 4,5). These results show that MLM prediction head potentially can be used as a translation layer to project query and corpus into the same language, which is also reported by Nair et al. [2022]. It is possible to leverage the transformers pre-trained with translation language modeling [Feng et al., 2022] and the established parallel training data [Bonifacio et al., 2021] to improve mAggretriever’s cross-lingual retrieval capability, which we leave for future work.

Comparisons to State-of-the-Art multilingual retrievers. Some previous state-of-the-art multilingual retrievers are evaluated on the test queries of Mr. TyDi [Zhang et al., 2021], the multilingual retrieval dataset similar to MIRACL but with sparse relevance judgements and less languages. In order to compare with the previous models, following the model inference settings in MIRACL, we evaluate our mAggretriever with the backbone of XLM-R. The full results are

Table 7.5: Mr. TyDi multilingual retrieval comparisons with state-of-the-art multilingual retrievers. Bold denotes the best effectiveness. multi CL denotes multilingual contrastive learning.

Mr. TyDi Test												
Models	multi CL	ar	bn	fi	id	ja	ko	ru	sw	te	th	avg.
		MRR@100										
(a) BM25 [Zhang et al., 2021]	✗	36.7	41.3	28.8	38.2	21.7	28.1	32.9	39.6	42.4	41.7	35.1
(b) mColBERT [Bonifacio et al., 2021]	✓	55.3	48.8	41.3	55.5	36.6	36.7	48.2	44.8	61.6	-	-
(c) mContriever [Izacard et al., 2022]	✓	43.4	42.3	35.1	42.6	32.4	34.2	36.1	51.2	37.4	40.2	39.5
(d) CCP [Wu et al., 2022]	✗	42.6	45.7	37.2	46.2	37.7	34.6	36.0	39.2	47.0	48.9	41.5
(e) MSM [Zhang et al., 2023a]	✗	51.6	53.0	39.4	50.5	32.0	36.8	37.2	43.4	62.6	53.5	44.7
(1) XLM-R _{CLS}	✗	41.9	40.8	27.8	39.9	32.5	33.0	27.7	23.7	54.2	46.1	36.8
(2) XLM-R _{AGG}	✗	52.3	55.8	43.2	55.0	40.4	40.5	41.5	45.1	77.5	57.3	50.8
(3) XLM-R _{AGG} ^{ig}	✗	52.3	55.2	43.0	54.8	41.1	40.4	44.9	46.0	76.2	58.7	51.2
(4) XLM-R _{AGG} ^{self}	✗	52.0	58.5	42.6	54.8	39.2	41.6	44.3	47.3	74.7	58.3	51.3

tabulated in Table 7.5. We still observe that all the variants of mAggretriever outperform previous state of the art in 6 out of 10 languages. Note that most of the previous retrievers undergo multilingual contrastive learning. For example, mContriever are pre-trained on the corpus with 29 languages while mColBERT are fine-tuned on multilingual MS MARCO dataset introduced by Bonifacio et al. [2021].

7.5 Summary

In this chapter, we introduce mAggretriever, an extension of Aggretriever for multilingual retrieval, by combining lexical and semantic features in pre-trained language models for multilingual dense retrieval. We propose target-language and self-MLM predictions to enhance the efficiency of mAggretriever. Our study highlights the efficiency advantage of self-MLM prediction in multilingual retrieval, while target-language MLM prediction offers flexibility for cross-lingual retrieval. Importantly, mAggretriever, solely fine-tuned on English data, demonstrates competitive multilingual retrieval capabilities compared to state-of-the-art dense retrievers.

Chapter 8

Conclusion and Future Work

In this thesis, we present a range of techniques to address the challenges of building robust retrieval systems using dense retrieval models. Specifically, we tackle two primary challenges: reducing the training cost of robust dense retrieval models and minimizing runtime costs when deploying advanced dense–sparse hybrid retrieval models. Our contributions are summarized as follows:

- We propose a novel knowledge distillation approach utilizing tightly coupled bi-encoders, which significantly reduces the computational cost of training dense retrieval models.
- Instead of relying on large-scale web-crawled training data, we propose a novel approach to automatically augment diverse training data from the MS MARCO passage corpus. Dense retrieval models trained on this augmented dataset demonstrate superior retrieval effectiveness across various domains and tasks.
- We propose a dense representation framework for semantic and lexical matching, which reduces runtime costs when deploying advanced hybrid retrieval methods, such as dense–sparse fusion and bi-encoder hybrid retrieval models.
- We propose a newly designed dense retrieval model that fully leverages semantic and lexical features from pre-trained transformers. This model demonstrates training efficiency and robustness across various domains, tasks, and languages.

Some promising future directions emerge from this thesis. For instance, the state of the art could be further advanced by fine-tuning our hybrid retrieval model (introduced in Chapter 5) using the diverse augmentation approach proposed in Chapter 4. Additionally, the diverse augmentation

approach could be extended to multilingual retrieval scenarios. We leave these exciting avenues for future exploration.

At the end, we discuss some promising directions to build robust retrieval models by leveraging the capability of large language models [LLMs; Brown et al., 2020, Touvron et al., 2023] and multimodal LLMs [MLLMs; Liu et al., 2023a, 2024, Dai et al., 2024] as follows:

- **Search Planning:** Information retrieval underpins a wide range of downstream tasks. In real-world scenarios, many queries involve multi-step retrieval [Yang et al., 2018], planning [Shao et al., 2024], and reasoning [Xiao et al., 2024]. For instance, answering a seemingly simple question like, “What is the stock price of the world’s most valuable company?” requires models to first identify the current most valuable company before retrieving its stock price. Similarly, composing a Wikipedia-style document on “Olympic Games Paris 2024” necessitates identifying related subtopics and conducting targeted searches for each. These tasks demand that retrieval systems go beyond simple search capabilities to include planning and reasoning. Integrating large language models (LLMs) into retrieval systems offers a promising pathway to addressing these challenges [Khattab et al., 2024, Asai et al., 2024].
- **Multimodal Retrieval:** Beyond complex text-based queries, user queries and documents often involve multimodal information, such as text combined with images [Liu et al., 2023b, Wei et al., 2023]. For example, given a multimodal query that includes an image of the White House alongside the text, “What is the history of the building in the image?”, the model must first identify the White House in the image and then integrate the query term “history” with the context “the White House” for accurate retrieval. Furthermore, documents may come in various formats, such as slides, PDFs, and tables [Tanaka et al., 2023, Ma et al., 2024, Faysse et al., 2024]. Leveraging multimodal large language models (MMLMs) to interpret and process multimodal information across these diverse formats represents a promising direction for developing retrieval systems capable of addressing broader and more versatile search scenarios.

References

- Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. On the cross-lingual transferability of monolingual representations. In *Proc. ACL*, pages 4623–4637, 2020.
- Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *Proc. ICLR*, 2024.
- Yang Bai, Xiaoguang Li, Gang Wang, Chaoliang Zhang, Lifeng Shang, Jun Xu, Zhaowei Wang, Fangshan Wang, and Qun Liu. SparTerm: Learning term-based sparse representation for fast text retrieval. *arXiv:2010.00768*, 2020.
- Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. MS MARCO: A human generated machine reading comprehension dataset. *arXiv:1611.09268*, 2016.
- Oren Barkan, Noam Razin, Itzik Malkiel, Ori Katz, Avi Caciularu, and Noam Koenigstein. Scalable attentive sentence-pair modeling via distilled sentence embedding. In *Proc. AAAI*, 2020.
- Luiz Henrique Bonifacio, Vitor Jeronymo, Hugo Queiroz Abonizio, Israel Campiotti, Marzieh Fadaee, Roberto Lotufo, and Rodrigo Nogueira. mMARCO: A multilingual version of MS MARCO passage ranking dataset. *arXiv:2108.13897*, 2021.
- Vera Boteva, Demian Gholipour, Artem Sokolov, and Stefan Riezler. A full-text learning to rank dataset for medical information retrieval. In *Proc. ECIR*, 2016.
- Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a ”siamese” time delay neural network. In *Proc. NeurIPS*, page 737–744, 1993.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel

- Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proc. NeurIPS*, pages 1877–1901, 2020.
- Wei-Cheng Chang, Felix X. Yu, Yin-Wen Chang, Yiming Yang, and Sanjiv Kumar. Pre-training tasks for embedding-based large-scale retrieval. In *Proc. ICLR*, 2020.
- Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proc. ICML*, 2020.
- Xilun Chen, Kushal Lakhotia, Barlas Oguz, Anchit Gupta, Patrick Lewis, Stan Peshterliev, Yashar Mehdad, Sonal Gupta, and Wen-tau Yih. Salient phrase aware dense retrieval: Can a dense retriever imitate a sparse one? In *Proc. EMNLP*, pages 250–262, 2022.
- Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel Weld. SPECTER: Document-level representation learning using citation-informed transformers. In *Proc. ACL*, pages 2270–2282, 2020.
- Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Un-supervised cross-lingual representation learning at scale. In *Proc. ACL*, pages 8440–8451, 2020.
- Nick Craswell, Bhaskar Mitra, and Daniel Campos. Overview of the TREC 2019 deep learning track. In *Proc. TREC*, 2019.
- Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. Overview of the TREC 2020 deep learning track. In *Proc. TREC*, 2020.
- Wenliang Dai, Nayeon Lee, Boxin Wang, Zhuoling Yang, Zihan Liu, Jon Barker, Tuomas Rintamäki, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. NVLM: Open frontier-class multimodal LLMs. *arXiv:2409.11402*, 2024.
- Zhuyun Dai and Jamie Callan. Context-aware term weighting for first stage passage retrieval. In *Proc. SIGIR*, page 1533–1536, 2020.
- Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proc. WSDM*, page 126–134, 2018.

- Zhuyun Dai, Vincent Y Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith Hall, and Ming-Wei Chang. Promptagator: Few-shot dense retrieval from 8 examples. In *Proc. ICLR*, 2023.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL*, pages 4171–4186, 2019.
- Thomas Diggelmann, Jordan Boyd-Graber, Jannis Bulian, Massimiliano Ciaramita, and Markus Leippold. CLIMATE-FEVER: A dataset for verification of real-world climate claims. *arXiv:2012.00614*, 2020.
- Zhe Dong, Jianmo Ni, Dan Bikel, Enrique Alfonseca, Yuan Wang, Chen Qu, and Imed Zitouni. Exploring dual encoder architectures for question answering. In *Proc. EMNLP*, pages 9414–9419, December 2022.
- Manuel Faysse, Hugues Sibille, Tony Wu, Bilel Omrani, Gautier Viaud, Céline Hudelot, and Pierre Colombo. ColPali: Efficient document retrieval with vision language models. *arXiv:2407.01449*, 2024.
- Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang. Language-agnostic BERT sentence embedding. In *Proc. ACL*, pages 878–891, 2022.
- Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. SPLADE v2: Sparse lexical and expansion model for information retrieval. *arXiv:2109.10086*, 2021a.
- Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. SPLADE: Sparse lexical and expansion model for first stage ranking. In *Proc. SIGIR*, page 2288–2292, 2021b.
- Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. From distillation to hard negative sampling: Making sparse neural IR models more effective. In *Proc. SIGIR*, page 2353–2359, 2022.
- Luyu Gao and Jamie Callan. Condenser: a pre-training architecture for dense retrieval. In *Proc. EMNLP*, pages 981–993, 2021.
- Luyu Gao and Jamie Callan. Unsupervised corpus aware language model pre-training for dense passage retrieval. In *Proc. ACL*, pages 2843–2853, 2022.
- Luyu Gao, Zhuyun Dai, and Jamie Callan. Understanding BERT rankers under distillation. In *Proc. ICTIR*, pages 149–152, 2020.

- Luyu Gao, Zhuyun Dai, and Jamie Callan. COIL: Revisit exact lexical match in information retrieval with contextualized inverted list. In *Proc. NAACL*, pages 3030–3042, 2021a.
- Luyu Gao, Zhuyun Dai, Tongfei Chen, Zhen Fan, Benjamin Van Durme, and Jamie Callan. Complement lexical retrieval model with semantic residual embeddings. In *Proc. ECIR*, pages 146–160, 2021b.
- Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Tevatron: An efficient and flexible toolkit for neural retrieval. In *Proc. SIGIR*, page 3120–3124, 2023.
- Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 744–755, 2014.
- Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *Proc. ICML*, 2020.
- Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proc. AISTATS*, pages 297–304, 2010.
- Faegheh Hasibi, Fedor Nikolaev, Chenyan Xiong, Krisztian Balog, Svein Erik Bratsberg, Alexander Kotov, and Jamie Callan. DBpedia-Entity v2: A test collection for entity search. In *Proc. SIGIR*, page 1265–1268, 2017.
- Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *Proc. NeurIPS: Deep Learning and Representation Learning Workshop*, 2015.
- Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. Improving efficient neural ranking models with cross-architecture knowledge distillation. *arXiv:2010.02666*, 2020.
- 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 *Proc. SIGIR*, page 113–122, 2021.
- Sebastian Hofstätter, Omar Khattab, Sophia Althammer, Mete Sertkan, and Allan Hanbury. Introducing neural bag of whole-words with colberter: Contextualized late interactions using enhanced reduction. In *Proc. CIKM*, page 737–747, 2022.
- Doris Hoogeveen, Karin M. Verspoor, and Timothy Baldwin. CQADupStack: A benchmark data set for community question-answering research. In *Proc. ADCS*, 2015.

- Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proc. CIKM*, page 2333–2338, 2013.
- Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. STOC*, page 604–613, New York, NY, USA, 1998.
- Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*, 2022.
- Kyoung-Rok Jang, Junmo Kang, Giwon Hong, Sung-Hyon Myaeng, Joohee Park, Taewon Yoon, and Heecheol Seo. Ultra-high dimensional sparse representations with binarization for efficient text retrieval. In *Proc. EMNLP*, pages 1016–1029, 2021.
- Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, pages 535–547, 2021.
- Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proc. ACL*, pages 1601–1611, 2017.
- Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 117–128, 2011.
- Ehsan Kamalloo, Xinyu Zhang, Odunayo Ogundepo, Nandan Thakur, David Alfonso-hermelo, Mehdi Rezagholizadeh, and Jimmy Lin. Evaluating embedding APIs for information retrieval. In *Proc. ACL (Volume 5: Industry Track)*, pages 518–526, 2023.
- Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proc. EMNLP*, pages 6769–6781, 2020.
- Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proc. SIGIR*, page 39–48, 2020.
- Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *Proc. ICLR*, 2024.

- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural Questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, pages 452–466, 2019.
- Carlos Lassance. Extending English IR methods to multi-lingual IR. *arXiv:2302.14723*, 2023.
- Carlos Lassance and Stephane Clinchant. The tale of two MSMARCO - and their unfair comparisons. In *Proc. SIGIR*, page 2431–2435, 2023.
- Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. In *Proc. ACL*, pages 6086–6096, 2019.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proc. NeurIPS*, pages 9459–9474, 2020.
- Minghan Li, Sheng-Chieh Lin, Barlas Oguz, Asish Ghoshal, Jimmy Lin, Yashar Mehdad, Wen-tau Yih, and Xilun Chen. CITADEL: Conditional token interaction via dynamic lexical routing for efficient and effective multi-vector retrieval. In *Proc. ACL*, pages 11891–11907, 2023.
- Minghan Li, Xilun Chen, Ari Holtzman, Beidi Chen, Jimmy Lin, Wen tau Yih, and Xi Victoria Lin. Nearest neighbor speculative decoding for llm generation and attribution, 2024.
- Jimmy Lin. A proposed conceptual framework for a representational approach to information retrieval. *SIGIR Forum*, 2022.
- Jimmy Lin and Xueguang Ma. A few brief notes on DeepImpact, COIL, and a conceptual framework for information retrieval techniques. *arXiv:2106.14807*, 2021.
- Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proc. SIGIR*, page 2356–2362, 2021a.
- Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. *Pretrained Transformers for Text Ranking: BERT and Beyond*. Morgan & Claypool, 2021b.
- Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. In-batch negatives for knowledge distillation with tightly-coupled teachers for dense retrieval. In *Proc. RepL4NLP*, pages 163–173, 2021c.

- Sheng-Chieh Lin, Akari Asai, Minghan Li, Barlas Oguz, Jimmy Lin, Yashar Mehdad, Wen-tau Yih, and Xilun Chen. How to Train your DRAGON: Diverse augmentation towards generalizable dense retrieval. In *Proc. Findings of EMNLP*, pages 6385–6400, 2023.
- Sheng-Chieh Lin, Luyu Gao, Barlas Oguz, Wenhan Xiong, Jimmy Lin, Wen tau Yih, and Xilun Chen. FLAME : Factuality-aware alignment for large language models. In *Proc. NeurIPS*, 2024.
- Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Proc. NeurIPS*, pages 34892–34916, 2023a.
- Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. LLaVA-NeXT: Improved reasoning, ocr, and world knowledge, January 2024. URL <https://llava-vl.github.io/blog/2024-01-30-llava-next/>.
- Ting Liu, Andrew W. Moore, Alexander Gray, and Ke Yang. An investigation of practical approximate nearest neighbor algorithms. In *Proc. NeurIPS*, page 825–832, 2004.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv:1907.11692*, 2019.
- Zhenghao Liu, Chenyan Xiong, Yuanhuiyi Lv, Zhiyuan Liu, and Ge Yu. Universal vision-language dense retrieval: Learning a unified representation space for multi-modal retrieval. In *Proc. ICLR*, 2023b.
- Shuqi Lu, Di He, Chenyan Xiong, Guolin Ke, Waleed Malik, Zhicheng Dou, Paul Bennett, Tie-Yan Liu, and Arnold Overwijk. Less is more: Pretrain a strong Siamese encoder for dense text retrieval using a weak decoder. In *Proc. EMNLP*, pages 2780–2791, November 2021.
- Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. Sparse, dense, and attentional representations for text retrieval. *Trans. Assoc. Comput. Linguistics*, pages 329–345, 2021.
- Xueguang Ma, Sheng-Chieh Lin, Minghan Li, Wenhui Chen, and Jimmy Lin. Unifying multi-modal retrieval via document screenshot embedding, 2024.
- Joel Mackenzie, Andrew Trotman, and Jimmy Lin. Efficient document-at-a-time and score-at-a-time query evaluation for learned sparse representations. *ACM Trans. Inf. Syst.*, 2023.

- Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. WWW’18 open challenge: Financial opinion mining and question answering. In *Companion Proceedings of the The Web Conference 2018*, page 1941–1942, 2018.
- Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 824–836, 2020a.
- Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836, 2020b.
- Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. Faster blockmax wand with variable-sized blocks. In *Proc. SIGIR*, page 625–634, 2017.
- Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. Learning passage impacts for inverted indexes. In *Proc. SIGIR*, page 1723–1727, 2021.
- Suraj Nair, Eugene Yang, Dawn Lawrie, James Mayfield, and Douglas W. Oard. Learning a sparse representation model for neural CLIR. In *Proc. DESIRES*, 2022.
- Anastasios Nentidis, Georgios Katsimpras, Anastasia Krithara, Salvador Lima López, Eulália Farré-Maduell, Luis Gasco, Martin Krallinger, and Georgios Paliouras. Overview of bioasq 2023: The eleventh bioasq challenge on large-scale biomedical semantic indexing and question answering. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, page 227–250, 2023.
- Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernandez Abrego, Ji Ma, Vincent Zhao, Yi Luan, Keith Hall, Ming-Wei Chang, and Yinfei Yang. Large dual encoders are generalizable retrievers. In *Proc. EMNLP*, pages 9844–9855, 2022.
- Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with BERT. *arXiv:1901.04085*, 2019.
- Rodrigo Nogueira and Jimmy Lin. From doc2query to docTTTTTquery, 2019.
- Barlas Oguz, Kushal Lakhotia, Anchit Gupta, Patrick Lewis, Vladimir Karpukhin, Aleksandra Piktus, Xilun Chen, Sebastian Riedel, Scott Yih, Sonal Gupta, and Yashar Mehdad. Domain-matched pre-training tasks for dense retrieval. In *Proc. NAACL*, pages 1524–1534, 2022.

- Biswajit Paria, Chih-Kuan Yeh, Ian E.H. Yen, Ning Xu, Pradeep Ravikumar, and Barnabás Póczos. Minimizing flops to learn efficient sparse representations. In *Proc. ICLR*, 2020.
- Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. *arxiv:2010.08191v1*, 2020.
- Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In *Proc. NAACL*, pages 5835–5847, June 2021.
- 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. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.
- Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proc. EMNLP*, pages 2383–2392, 2016.
- Ori Ram, Gal Shachaf, Omer Levy, Jonathan Berant, and Amir Globerson. Learning to retrieve passages without supervision. In *Proc. NAACL*, pages 2687–2700, 2022.
- Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proc. EMNLP*, pages 3982–3992, November 2019.
- Nils Reimers and Iryna Gurevych. Making monolingual sentence embeddings multilingual using knowledge distillation. In *Proc. EMNLP*, pages 4512–4525, 2020.
- Ruiyang Ren, Yingqi Qu, Jing Liu, Wayne Xin Zhao, QiaoQiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. RocketQAv2: A joint training method for dense passage retrieval and passage re-ranking. In *Proc. EMNLP*, pages 2825–2835, November 2021.
- Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at TREC-3. In *Proc. TREC*, 1994.
- Uma Roy, Noah Constant, Rami Al-Rfou, Aditya Barua, Aaron Phillips, and Yinfei Yang. LAReQA: Language-agnostic answer retrieval from a multilingual pool. In *Proc. EMNLP*, pages 5919–5930, 2020.
- Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv:1910.01108*, 2019.

- Keshav Santhanam, Omar Khattab, Christopher Potts, and Matei Zaharia. PLAID: An efficient engine for late interaction retrieval. In *Proc. CIKM*, page 1747–1756, 2022a.
- Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proc. NAACL*, pages 3715–3734, 2022b.
- Christopher Sciavolino, Zexuan Zhong, Jinhyuk Lee, and Danqi Chen. Simple entity-centric questions challenge dense retrievers. In *Proc. EMNLP*, 2021.
- Yijia Shao, Yucheng Jiang, Theodore Kanell, Peter Xu, Omar Khattab, and Monica Lam. Assisting in writing Wikipedia-like articles from scratch with large language models. In *Proc. NAACL*, pages 6252–6278, 2024.
- Tao Shen, Xiubo Geng, Chongyang Tao, Can Xu, Guodong Long, Kai Zhang, and Daxin Jiang. Unifire: A unified retriever for large-scale retrieval. In *Proc. KDD*, page 4787–4799, 2023.
- Ian Soboroff, Shudong Huang, and Donna Harman. Trec 2019 news track overview. In *Proc. TREC*, 2020.
- Axel Suarez, Dyaa Albakour, David Corney, Miguel Martinez, and José Esquivel. A data collection for evaluating the retrieval of related tweets to news articles. In *Proc. ECIR*, 2018.
- Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. ERNIE 2.0: A continual pre-training framework for language understanding. *arXiv:1907.12412*, 2019.
- Ryota Tanaka, Kyosuke Nishida, Kosuke Nishida, Taku Hasegawa, Itsumi Saito, and Kuniko Saito. SlideVQA: a dataset for document visual question answering on multiple images. In *Proc. AAAI*, 2023.
- Nandan Thakur, Nils Reimers, Johannes Daxenberger, and Iryna Gurevych. Augmented SBERT: Data augmentation method for improving bi-encoders for pairwise sentence scoring tasks. In *Proc. NAACL*, pages 296–310, 2021a.
- Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Proc. NeurIPS*, 2021b.
- Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. *Efficient Query Processing for Scalable Web Search*. Now Foundations and Trends, 2018.

- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*, 2023.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. NeurIPS*, pages 5998–6008, 2017.
- Ellen Voorhees. Overview of the trec 2004 robust retrieval track. In *Proc. TREC*, 2005.
- Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R. Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. TREC-COVID: constructing a pandemic information retrieval test collection. *SIGIR Forum*, 2021.
- Henning Wachsmuth, Shahbaz Syed, and Benno Stein. Retrieval of the best counterargument without prior topic knowledge. In *Proc. ACL*, pages 241–251, 2018a.
- Henning Wachsmuth, Shahbaz Syed, and Benno Stein. Retrieval of the best counterargument without prior topic knowledge. In *Proc. ACL*, pages 241–251, 2018b.
- David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. Fact or fiction: Verifying scientific claims. In *Proc. EMNLP*, pages 7534–7550, 2020.
- Kexin Wang, Nandan Thakur, Nils Reimers, and Iryna Gurevych. GPL: Generative pseudo labeling for unsupervised domain adaptation of dense retrieval. In *Proc. NAACL*, pages 2345–2360, 2022.
- Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv:2212.03533*, 2024.

- Cong Wei, Yang Chen, Haonan Chen, Hexiang Hu, Ge Zhang, Jie Fu, Alan Ritter, and Wenhui Chen. UniIR: Training and benchmarking universal multimodal information retrievers. *arXiv:2311.17136*, 2023.
- Ning Wu, Yaobo Liang, Houxing Ren, Linjun Shou, Nan Duan, Ming Gong, and Daxin Jiang. Unsupervised context aware sentence representation pretraining for multi-lingual dense retrieval. *arXiv:2206.03281*, 2022.
- Xing Wu, Guangyuan Ma, Meng Lin, Zijia Lin, Zhongyuan Wang, and Songlin Hu. Contextual masked auto-encoder for dense passage retrieval. In *Proc. AAAI*, 2023.
- Chenghao Xiao, G Thomas Hudson, and Noura Al Moubayed. RAR-b: Reasoning as retrieval benchmark. *arXiv:2404.06347*, 2024.
- Shitao Xiao, Zheng Liu, Yingxia Shao, and Zhao Cao. RetroMAE: Pre-training retrieval-oriented language models via masked auto-encoder. In *Proc. EMNLP*, pages 538–548, 2022.
- Ji Xin, Chenyan Xiong, Ashwin Srinivasan, Ankita Sharma, Damien Jose, and Paul Bennett. Zero-shot dense retrieval with momentum adversarial domain invariant representations. In *Proc. ACL*, pages 4008–4020, 2022.
- Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In *Proc. SIGIR*, page 55–64, 2017.
- Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *Proc. ICLR*, 2021.
- Ikuya Yamada, Akari Asai, and Hannaneh Hajishirzi. Efficient passage retrieval with hashing for open-domain question answering. In *Proc. ACL*, pages 979–986, 2021.
- Jheng-Hong Yang, Xueguang Ma, and Jimmy Lin. Sparsifying sparse representations for passage retrieval by top- k masking. *arXiv:2112.09628*, 2021.
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proc. EMNLP*, pages 2369–2380, 2018.
- Yue Yu, Chenyan Xiong, Si Sun, Chao Zhang, and Arnold Overwijk. COCO-DR: Combating distribution shift in zero-shot dense retrieval with contrastive and distributionally robust learning. In *Proc. EMNLP*, pages 1462–1479, 2022.

- Hamed Zamani, Mostafa Dehghani, W. Bruce Croft, Erik Learned-Miller, and Jaap Kamps. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *Proc. CIKM*, page 497–506, 2018.
- Hansi Zeng, Hamed Zamani, and Vishwa Vinay. Curriculum learning for dense retrieval distillation. In *Proc. SIGIR*, page 1979–1983, 2022.
- Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. Learning to retrieve: How to train a dense retrieval model effectively and efficiently. *arXiv:2010.10469*, 2020a.
- Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. RepBERT: Contextualized text embeddings for first-stage retrieval. *arXiv:2006.15498*, 2020b.
- Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Jointly optimizing query encoder and product quantization to improve retrieval performance. In *Proc. CIKM*, page 2487–2496, 2021a.
- Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Optimizing dense retrieval model training with hard negatives. In *Proc. SIGIR*, page 1503–1512, 2021b.
- Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Learning discrete representations via constrained clustering for effective and efficient dense retrieval. In *Proc. WSDM*, page 1328–1336, 2022.
- Shunyu Zhang, Yaobo Liang, Ming Gong, Daxin Jiang, and Nan Duan. Modeling sequential sentence relation to improve cross-lingual dense retrieval. In *Proc. ICLR*, 2023a.
- Xinyu Zhang, Xueguang Ma, Peng Shi, and Jimmy Lin. Mr. TyDi: A multi-lingual benchmark for dense retrieval. In *Proc. Workshop on Multilingual Representation Learning*, pages 127–137, 2021.
- Xinyu Zhang, Kelechi Ogueji, Xueguang Ma, and Jimmy Lin. Toward best practices for training multilingual dense retrieval models. *ACM Trans. Inf. Syst.*, 2023b.
- Xinyu Zhang, Nandan Thakur, Odunayo Ogundepo, Ehsan Kamalloo, David Alfonso-Hermelo, Xiaoguang Li, Qun Liu, Mehdi Rezagholizadeh, and Jimmy Lin. MIRACL: A Multilingual Retrieval Dataset Covering 18 Diverse Languages. *Transactions of the Association for Computational Linguistics*, pages 1114–1131, 2023c.
- Shengyao Zhuang and Guido Zuccon. TILDE: Term independent likelihood model for passage re-ranking. In *Proc. SIGIR*, page 1483–1492, 2021a.

Shengyao Zhuang and Guido Zuccon. Fast passage re-ranking with contextualized exact term matching and efficient passage expansion. *arXiv:2108.08513*, 2021b.